Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/15BE07Vuckovic

AUTOMATIZOVANO TESTIRANJE KORISNOSTI VEB APLIKACIJA POMOĆU JASMIN OKVIRA

AUTOMATIC TESTING THE USEFULNESS OF WEB APPLICATIONS USING THE JASMINE FRAMEWORK

Marko Vučković, *Fakultet tehničkih nauka, Novi Sad*

Oblast – Računarstvo i automatika

Kratak sadržaj – *Rad se sastoji iz dva dela. Prvi deo su osnove testiranja, a zatim i primena istog uz pomoć Jasmin okvira. Jasmin je jedan od alata koji se mogu primeniti za testiranje veb aplikacija. Prikazane su osnovne funckionalnosti koje Jasmin sa sobom donosi*

Ključne reči: *Jasmin, testiranje, veb aplikacije*

Abstract – *This paper consists of two parts. The first part is the basics of testing, and second part is testing of web applications using help of Jasmine framework. Jasmine is one of tools that can be used for testing web applications. The basic functionalities that Jasmine brings are presented in the paper.*

Keywords: *Jasmine, testing, web applications.*

1. UVOD

U poslednjoj deceniji svedoci smo da je softver pojam koji nas svakodnevno okružuje. Gde god da krenemo, bilo to u prodavnicu, na sport ili negde drugde, sve je postalo kompjuterizovano. Softverski sistemi postaju sve više važniji, kompleksniji i raste značaj njihovog kvaliteta. Samim tim, kontrola kvaliteta softvera je važna koliko i sama izrada rešenja. Propust može dovesti do ozbiljnjih problema, ukoliko se govori o oblastima poput medicine, ili vazduhoplostva, dok na drugoj strani može izazvati nepoverenje klijenata, a poverenje u softver je jedno od najbitnijih stavki kada se softver isporuči klijentu. Testiranje softvera predstavlja samo jedan deo procesa kontrole kvaliteta softvera.

Različite metode koriste se za testiranje različitih slojeva softvera, a ono što je svima zajedničko jeste da se mogu izvoditi manuelno ili automatski korišćenjem posebnih alata. I jedan i drugi način imaju svoje prednosti i mane, ali se u poslednje vreme sve više teži automatizaciji jer se automatizacijom testiranja skraćuje vreme potrebno za testiranje.

Jasmine je okvir za automatizovano testiranje web aplikacija. Jedan je od najčešćih izbora stručnjaka koji se bave testiranjem softvera. Univerzalan je, lak za korišćenje, i podržava testiranje u većinih dostupnih pretraživača. U ovom radu su opisane osnovne funkcionalnosti koje dolaze uz Jasmin okvir.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio prof. dr Dragan Ivetić.

2. TESTIRANJE SOFTVERA

Testiranje softvera je proces koji se koristi da bi se utvrdila ispravnost, potpunost i kvalitet razvijenog softvera u potpunosti da utvrdi ispravnost računarskog softvera. Samo proces formalne verifikacije može da pokaže da u softveru nema grešaka. Testiranje softvera je način da se obezbedi manji broj grešaka, manji trošak održavanja i sveukupne cene softvera.

2.1. Osnovni pojmovi

Često se dešava u praksi da se određeni pojmovi pogrešno koriste i tumače. Koja je razlika između greške i defekta? Šta je incident? Kako bi se izbegle nedoumice, neophodno je na samom početku definisati osnovne pojmove. Greška jeste rezultat ljudske aktivnosti, bilo za vreme specifikacije programa ili tokom pisanja programa. Na primer, neka funkcionalnost može biti pogrešno specificirana, što će kasnije dovesti do pogrešne implementacije, ili se može napraviti neka greška u kodiranju, koja će dovesti do neispravnog rada programa. Posledica greške se naziva mana ili defekt – programu nešto nedostaje , ili ima funkciju koja se ne ponaša ispravno. Otkaz nastaje kada sistem nije u mogućnosti da obavi funkciju koju korisnik od njega zahteva, jer se aktivira i izvršava defektni kod. Kada postoji defekt u softveru, javlja se simptom kojim korisnik postaje svestan otkaza u sistemu.

Ovaj simptom se naziva incident (ili poremećaj). Testiranje je postupak izvršavanja softvera sa testovima. Testiranje može imati jedan od dva osnovna cilja: da se pronađu otkazi ili da se demonstrira ispravno ponašanje sistema. Testiranjem se mogu otkriti otkazi u sistemu, koje je kasnije potrebno ispraviti.

Test je skup ulaznih vrednosti, preduslova izvršavanja, očekivanih rezultata i stanja u kome sistem treba da ostane nakon završetka. Test se razvija sa ciljem da ispita određeno ponašanje programa, na primer da izvrši određenu putanju kroz program ili da verifikuje da li je određeni zahtev ispravno implementiran. Testiranje softvera se može definisati kao proces izvršavanja programa sa ciljem da se pronađu greške. Ukoliko posmatramo širi smisao, testiranje je proces koji se sastoji od statičkih i dinamičkih aktivnosti, sa ciljem da se odredi da li softver zadovoljava specificirane zahteve i otkriju defekti ili da se demostrira ispravan rad.

Testiranje softvera je deo procesa osiguravanja kvaliteta (engl. Quality Assurance), čiji je zadatak da se obezbedi isporuka kvalitetnog softvera u zahtevanom vremenskom roku. Test set se odnosi na skup svih testova koje je

potrebno izvršiti nad softverom. Testovi se ne biraju nasumično, već je neophodno uložiti trud u pažljivo planiranje i dizajn. Čitava jedna faza u procesu testiranja softvera je odvojena u ovu svrhu. Nasumično odabrani test nema nikakvog značaja ukoliko detektuje grešku koja je već pronađena nekim drugim testom. Testiranje svih mogućih kombinacija ulaznih podataka ili takozvano iscrpno testiranje je nemoguće ili u najboljem slučaju nepraktično, pošto svaki netrivijalni sistem ima veoma veliki domen ulaznih podataka. Ukupan broj testova u test setu ne garantuje da će i testiranje biti uspešno.

2.2. Proces testiranja softvera

Testiranje je proces određivanja nivoa kvaliteta softvera. Nivo kvaliteta softvera i zahtevi variraju od sistema do sistema, ali neke osnovne karakteristike koje svi oni treba da poseduju i koje se vrednuju su: pouzdanost, stabilnost, prenosivost, kompatibilnost i upotrebljivost. Testiranje je usko povezano sa pojmovima verifikacije i validacije softvera.

Verifikacija je proces procene softvera ili njegovih komponenti sa ciljem da se utvrdi da li proizvodi određenih faza razvoja zadovljavaju uslove sa početka tih faza. Ovim procesom se utvrđuje koliko je proizvod jedne faze kompatibilan sa proizvodom prethodnih faza i da li svi delovi sistema, pojedinačno i kao celina, rade ispravno. Validacija se definiše kao proces procene softvera ili njegovih komponenti, tokom ili na kraju njegovog razvoja, sa ciljem da se utvrdi da li zadovljava polazne zahteve. Cilj validacije jeste da odgovori na pitanje da li se razvija pravi softver, to jest, softver koji je u skladu sa zahtevima korisnika definisanim na početku procesa razvoja i da li je takav softver upotrebljiv za korisnika. Validaciju obavljaju testeri kroz izvršavanje test skriptova.

2.3. Osnovni principi testiranja

Postoji sedam osnovnih principa testiranja, koji su se izdvojili kroz nekoliko decenija prakse testiranja softvera, i koji važe za bilo koji tip softvera. Posmatraju se kao smernice koje bi trebalo ispoštovati u svakom projektu, jer su se kroz istoriju pokazali kao tačni.

Principi glase: testiranje pokazuje prisustvo defekata, iscrpno testiranje nije moguće, rano testiranje, grupisanje defekata, paradoks pesticida, testiranje zavisi od konteksta, odsustvo grešaka ne garantuje da sistem radi kako treba.

2.4. Stragetije testiranja softvera

U odnosu na način na koji se posmatra sistem koji se testira, postoje dva pristupa testiranju – tehnika bele kutije i tehnika crne. Metoda crne kutije je oblik testiranja softvera gde unutrašnja struktura, dizajn i implementacija softvera ili softverskih jedinica koje se testiraju nisu poznate testeru. Kako implementacija i struktura softvera nije poznata, softver se posmatra kao crna kutija po čemu je ovaj metod i dobio ime. Jedina poznata informacija koju tester ima za određivanje i dizajn testova je specifikacija zahteva programa.

Metodom crne kutije se mogu otkriti nepostojeće funkcionalnosti ili pogrešne implementacije u softveru, greške u ponašanju softvera kao i problemi sa performansama sistema. Tehnika crne kutije se može primeniti na svim nivoima testiranja (jediničnom, integracionom i sistemskom nivou). Uticaj nivoa testiranja na metodu crne kutije se ogleda samo u kompleksnosti izvršavanja ove metode. Strukturno testiranje, poznato kao metoda bele ili staklene kutije, je tehnika testiranja gde je testerima poznata implementacija softvera. Za razliku od modela crne kutije koji je fukusiran na to šta softver radi, model bele kutije je fokusiran na to kako softver radi.

Cilj testiranja ovim modelom jeste da se izvrše sve programske strukture i sve strukture podataka u softveru. Ovde se proverava samo kod, a ne specifikacija. Ovaj metod se može primeniti na svim nivoima testiranja, ali se najčešće primenjuje nakon metode crne kutije. Metoda bele kutije koriste i sami developeri, jer se očekuje da su implementirane komponente temeljno testirane pre nego se integrišu u softver, odnosno daju testerima na detaljno testiranje.

Pokrivenost testiranja (eng. coverage) je mera do koje je neki softver ili određena softverska komponenta istestirana nekim skupom testova, u smislu procenta obuhvaćenih stavki. U slučaju da pokrivenost nije 100%, potrebno je proširiti skup testova s novim testovima. Pokrivenost se povećava tako da se sistemski pišu novi testovi da bi se pokrili svi delovi softvera, odnosno da se svi delovi softvera izvrše bar jednom.

2.5. Nivoi testiranja softvera

Testiranje softvera, posebno velikih sistema, vrši se na više nivoa, od kojih svaki ima specifične ciljeve testiranja. Najčešći nivoi su: jedinično testiranje, integraciono testiranje, sistemsko testiranje i testiranje prihvatljivosti. Jedinično testiranje je testiranje funkcionalnosti programskih komponeti (jedinica) nezavisno od ostalih delova sistema. Jedinicama programa se smatraju funkcije ili klase. Programer koji piše kod obično i izvodi jedinično testiranje svog dela koda u razvojnom okruženju koristeći neki od okvira za jedinično testiranje.

Cilj ovakvog načina testiranja jeste da se detektuju funkcionalne i strukturne greške u odgovarajućoj jedinici, koje se odmah po nalaženju i poprave. Faza integracionog testiranja dolazi na kraju jediničnog testiranja. Integraciono testiranje je u praksi često zanemareno i ne obavi se na adekvatan način što dovodi do ozbiljnih problema. Integraciono testiranje je faza u kojoj se pojedinačni modeli i jedinice spajaju i testiraju zajedno kao celina.

Komponente koje ulaze u integraciono testiranje su već prošle jedinično testiranje. Ove komponente se grupišu i nad njima se izvršavaju posebno pripremljeni testovi. Ovo testiranje je važno iz razloga da se izbegnu svi potencijalni problemi koji mogu nastati prilikom spajanja komponenti u celinu.

Sistemsko testiranje je testiranje sistema kao celine, nakon što se uspešno završi kompletna integracija. Svrha sistemskog testiranja je verifikacija da softver kao celina ispunjava zahteve koji su definisani u specifikaciji zahteva sistema. Sistemsko testiranje obuhvata testove koji se definišu na osnovu specifikacije zahteva sistema i obično ga obavljaju najiskusniji testeri.

Ponekad se za ovaj zadatak angažuje nezavisan tim testera, da bi se obezbedila potpuna objektivnost pri sistemskom testiranju. Testiranje prihvatljivosti je faza koja se obavlja kada se softver pravi za određenu namenu i određenog klijenta.

Kod ovog nivoa testiranja sam klijent daje sud o tome da li su ispunjeni svi zahtevi i da li isporučeni softver obavlja traženu svrhu. Često uključuje alfa i beta testiranje. Alfa testiranje se vrši tokom samog razvoja programa kad se programerima odmah ukazuje na greške. Beta testiranje se vrši onda kada su sve moguće softverske greške nađene i popravljene i obavlja se u realnom okruženju u kome softver i treba da se koristi.

2.6. Automatsko i manuelno testiranje softvera

Manuelno testiranje podrazumeva ručno izvršavanje test skritpova različitim alatima u kojima se prate koraci testa i beleže test rezultati. Izvršavanje testa se smatra uspešnim ako je ponašanje sistema koji se testira u skladu sa očekivanim ponašanjem.

Nasuprot manuelnom je automatsko testiranje koje zahteva postojanje određenog koda koji je napisan da bi se automatizovali koraci pri izvršavanju određenog test skirpta. Manuelno i automatsko testiranje prate iste faze procesa testiranja i mogu se primeniti na različitim novima i tipovima testiranja.

2.7. Tipovi testiranja softvera

U tipove testiranja softvera spadaju: testiranje performansi sistema, test prihvatanja od strane korisnika (UAT), regresivno testiranje, statičko testiranje, smoke testing i sanity testiranje. Performanse obuhvataju vreme odaziva softvera, pouzdanost, upotreba resursa i skalabilnost. Testiranje performansi je tip testiranja koji verifikuje da se sistemski softver ponaša na odgovarajući način pod nekim očekivanim opterećenjem.

Cilj testiranja performansi nije pronalazak novih defekata, već da se eliminišu potencijalni problemi koji utiču na performanse sistema. Test prihvatanja od strane korisnika (eng. User acceptance testing ili Acceptance testing) je testiranje koje vrši klijent kada je sistem spreman za isporuku, nakon što se ispravila većina defekata pronađena u sistemskoj fazi testiranja.

Cilj ovog testiranja jeste da klijent stekne poverenje u sistem koji je implementiran. Proveravaju se funkcionalnosti prema specifikaciji zahteva i određuje se da li sistem ispunjava potrebe krajnjih korisnika. Regresivno testiranje je tip testiranja softvera čiji je cilj da potvrdi da nove promene u kodu softvera nisu uzrokovale greške u već postojećim funkcionalnostima sistema.

Ovaj tip testiranja podrazumeva ponavljanje testova koji su već izvršeni u prethodnim iteracijama, da bi bili sigurni da postojeće funkcionalnosti rade ispravno i nakon implementacije novih komponenti.

Statičko testiranje je skup tehnika kojim se poboljšava kvalitet softvera, kao i efikasnost i produktivnost samog procesa razvoja softvera. Ovaj tip testiranja se bazira na procesu statičkog pregleda (eng. review), s ciljem da se defekti pronađu što ranije u procesu razvoja softvera.

Smoke test je oblik testiranja softvera koji proverava da li osnovne funkcionalnosti softvera rade na unapred određen način. Ovaj oblik testiranja nije detaljan, niti ulazi u dubinu funkcionalnosti.

Izvršava se mali broj testova samo da bi se proverilo da li osnovne funkcije softvera rade. Ovaj oblik testiranja se koristi kao osnovna provera da li je softver dovoljno stabilan i spreman za dalje testiranje. Test zdravog razuma (eng. Sanity test) je sličan smoke testu. Ovi termini se često mešaju u testiranju softvera. Iako razlike između ova dva tipa nisu velike, one ipak postoje. Smoke test se koristi za testiranje build-a softvera da bi se verifikovao ispravan rad osnovnih funkcionalnosti i izvršava se pre bilo kojeg detaljnog testiranja, kako testeri ne bi gubili vreme u slučaju nekih kritičnih problema. Sanity test se izvršava nad relativno stabilnim softverom. Ovaj tip testiranja vrše testeri nakon primanja build-a u kome postoje izmene u kodu ili je dodata nova funkcionalnost.

3. JASMIN OKVIR

Jasmin (eng. Jasmine) je razvojni okvir vodjen ponašanjem koji omogućuje pisanje testova za testiranje koda napisanog u JavaScript programskom jeziku. Napravljen je tako da ne zavisi od drugih JavaScript razvojnih okvira i za njegovo korišćenje nije potreban objektni model dokumenta (eng. Document Object Model - DOM). Ima jednostavnu i veoma razumljivu sintaksu tako da je relativno jednostavan za korišćenje čak i za programere koji nemaju iskustva sa pisanjem testova.

Kako je ovo razvojni okvir namenjen za testiranje koda, Jasmin se može koristiti kao alat komandne linije koji automatski pronalazi napisane testove, pokreće ih i izveštava o rezultatima.

3.1. Razvoj softvera vođen ponašanjem

U softverskom inženjerstvu, razvoj zasnovan na ponašanju je agilni process razvoja softvera koji podstiče saradnju među programerima, testerima i kupcima u okviru softverskog projekta. On ohrabruje timove da koriste razgovor i konkretne primere za formalizovanje zajedničkog razumevanja o tome kako se aplikacija treba ponašati. Nastao je iz ravoja zasnovanog na testovima (eng. Test-driven devolopment). Razvoj vođen ponašanjem kombinuje opšte tehnike i principe razvoja zasnovanog na testovima sa idejama iz objektno-orijentisane analize i dizajna kako bi se omogućili programerima, testerima i ostalima koji učestvuju u razvoju softvera zajednički alati i zajednički proces za saradnju na razvoju softvera.

3.2. Jasmin funkcionalnosti

Postoji mnogo funkcionalnosti koje Jasmin donosi sa sobom, a u ovoj sekciji će biti nabrojane neki od njih. Osim ugrađenih funkcionalnosti, postoji i mogućnost implentiranja prilagođenih za potrebe određenog projekta. Možda i najjednostavnija metoda je toEqual. Ona, kao što i sam naziv kaže, proverava da li je ono što se poredi isto, ali ne nužno da li je i objekat isti.

Metoda toBe provera da li je ono što je poređeno pripada istom objektu, dok toEqual proverava samo običnu jednakost. Da bismo testirali da li je nešto tačno ili netačno, koristimo metode toBeTruthy i toBeFalsy. Treba napomenuti da je Jasminova evaluacija identična evaluaciji Javascript-a kada je reč o tačnosti ili netačnosti. Ovo znači da je true uvek tačno, kao i string „pas“ ili broj 9, ili bilo koji objekat.

Ukoliko želimo da negiramo neku metodu, poput metode toEqual ili metode toContain, možemo jednostavno iskoristiti prefiks .not. Ponekad je potrebno proveriti da li je element prisutan u okviru liste, ili u okviru string-a. Tada je moguće iskoristi metodu toContain. Pored već navedenih, postoji još mnoštvo metoda koje Jasmin donosi sa sobom, a

neke od njih su sledeće: toBeDefined i toBeUndefined (provera da li je ono što se proverava definisano ili nije), toBeNull (provera da li je prosleđena vrednost null), toBeNaN (provera da li prosleđena vrednost nije broj), toBeGreatherThan, toBeLessThan (komparacija prosleđenih vrednosti, koja ne radi nužno samo sa brojevima), toBeCloseTo (provera da li je broj dovoljno blizu drugom prosleđenom broju), toMatch(provera da li prosleđena vrednost odgovara određenom izrazu).

Postoje trenuci kada za testiranje nisu dovoljne već postojeće metoda Jasmin okvira. Tada se mogu kreirati prilagođene metode. Da bi se to uradilo, mora se na početku svakog testa dodati željena metoda. Još jedna od korisnih funkcionalnosti Jasmin okvira su metode BeforeEach i AfterEach. One omogućavaju da se izvrše određeni delovi koda, pre ili posle svakog testa. Ova funkcionalnost dosta pomaže da bi se postigao čistiji kod, kao i čišćenje varijabli nakon samog testa.

Kada kod postane dosta kompleksniji, samim tim postoji i mnogo više testova i oni su sve kompleksniji. Tada se mogu formirati grupe, podgrupe, podgrupe podgrupa, itd. Ovo je moguće izvesti tako što se describe blok, koji služi za opis testa, stavi unutar drugog describe okvira.

3.3. Jasmin okvir – testiranje Angular aplikacije

TestBed dolazi sa modulom koji je konfigurisan kao i svi ostali moduli u okviru Angular aplikacije. Moguće je deklarisati komponente, direktive, servise, kao i importovati druge module. Ono što je bitno kod TestBed-a, on dolazi sa metodom configureTestingModule koja prima definicije modula koji želimo da testiramo. Nakon što je konfigurisana komponenta koja će biti testirana, potrebno je renderovati komponentu, kako bi se moglo pristupati njenim delovima.

TestBed poseduje metodu createComponent, povratna vrednost ove metode je fixture, instanca klase ComponentFixture. Fixture sadrži komponentu u sebi i pruža interfejs i za instancu komponente i za renderovani DOM.

Moduli su centralni deo svake Angular aplikacije. Ipak, teški su za testiranje jer u njima neke tipične logike, već uglavnom samo konfiguracija. Moduli jesu klase, ali su one same po sebi najčeće prazne. Glavni deo modula se nalazi u delu označenim sa NgModule, jer se tu nalaze metapodaci (deklaracije, ulazne i izlazne komponente, provajderi, itd..). Ipak, u toku vremena, može doći do grešaka i u okviru modula, pa je potrebno napisati testove i za module. Ove greške mogu biti uhvaćene ranije pomenutim smoke testovima.

Testiranje običnih servisa je poprilično jednostavno, mogu se testirati njihove metode, ali ono što je veoma važno, potrebno je testirati servise koji komunicaraju sa backend-om aplikacije, i šalju HTTP zahteve. Prvi korak testa, nakon onih već poznatih, je da se pozove metoda nad servisom koja šalje HTTP zahtev, u ovom slučaju je to metoda getByCode koja služi da vrati valutu koja odgovara prosleđenom stringu od 3 karaktera.

U drugom koraku je korišćen HttpTestingController. Ovaj kontroler poseduje metode za nalaženje zahteva po različitim kriterijumima.

Najjednostavnija od njih je ona koja je korišćena, metoda expectOne, koja očekuje da nađe tačno jedan zahtev za zadati kriterijum.

Ova metoda vraća instancu klase TestRequest, a ukoliko ne pronađe nijedan odgovarajući zahtev, metoda expectOne baci grešku.

Treći korak je formiranje odgovora servera sa lažnim podacima. U ovom slučaju je to linija koda koja nad zahtevom poziva metodu flush. Uloga ove metoda jeste da simulira uspešni “200 OK“ odgovor servera.

Četvrti korak, pre same provere rezultata testa, je poziv metode verify nad kontrolerom. Uloga ove metode je provera da li je preostalo zahteva koji čekaju. U ovom slučaju je pronađen jedan zahtev metodom expectOne i na njega je odgovoreno metodom flush.

4. ANALIZA USPEŠNOSTI JASMIN ALATA

Jasmin alat je jedan od najkorišćenijih alata kada je reč o alatima kojima se testiraju web aplikacije, ali pored njega, još se koriste i sledeći alati: Jest, Mocha, Cypress, Puppeteer, AVA, StoryBook, Enzyme, itd.. Svi alati imaju svoje mane i prednosti. Ono što je potrebno uraditi je pažljivo odabrati alati koji bi trebalo koristiti. Najpopularniji alati za testiranje su Jest, Jasmin i Mocha. Jest okvir za testiranje jeste verovatno najbolji izbor kada se radi o web aplikacijama koje su pisane u React framework-u.

Razlog za to je prilično jednostavan, Jest u kombinaciji sa React-om donosi zgodnu osobinu, a to su regresivni vizuelni testovi kojima se lako otkrivaju slučajne greške nastale na korisničkom interfejsu. Jest ima opciju da beleži slike ekrana(eng. screenshots) i na taj način može da poredi jednu te istu komponentu tokom njene dorade. Takođe zahteva minimalnu konfiguraciju i veoma je dobro dokumentovan. Ono što je njegova najveća mana u odnosu na Jasmin, je ta što u poređenju sa Jasmin testnim alatom Jest ne podržava ni blizu toliko biblioteka i alata koje nekada mogu biti veoma korisne.

Popularnost Jest-a jeste propraćena povećanim korišćenjem React framework-a, ali se svakako mora reći da uz Jasmin testni okvir spada u najčeće korišćene i čini njegove korisnike zadovoljnim. Mocha je do pre nekoliko godina bila možda i najpopularniji alat za testiranje, ali se pojavom Jest-a, Jasmin i Cypress alata njena upotreba znatno smanjila. Ono što je najveća mana ovog alata je vreme koje potrebno za podešavanje i konfiguraciju. Takođe, generalne perfomanse nisu na najvišem nivou u poređenju sa ostalim alatima.

Cypress testni alat donosi sa sobom nekoliko zgodnih osobina u odnosu na druge testne alate, kao što su: mogućnost automatskog skrola, omogućava direktne promene DOM-a, čuva slike ekrana na svakom koraku, što omogućava programeru da proveri stanje na svakom koraku, itd..

Najveća mana ovog testnog alata jeste to što podržava samo JavaScript framework za pisanje testova i to što ne može da rukuje sa više otvorenih tabova u internet pregledaču, niti sa više otvorenih pregledača.

Jasmin okvir za testiranje ipak, uz Jest, donosi najviše a pritom ima najmanje mana. Kompatibilan je sa svakim framework-om, što ga čini najfleksibilnijim alatom. Podrška za Jasmin alat se može naći na mnogo mesta u formama biblioteka, blogova i video tutorijala. Jasmin nudi elegantna rešenja i paterne.

Dostupan je u svim internet pregledačima. Trenutno dva verovatno najpopularnija frontend framework-a su svakako React i Angular. Kada je reč o Angular-u, svakako najbolji izbor za testiranje jeste Jasmin okvir, dok je Jest ipak izbor kada se radi o React framework-u, zbog prethodno navedenih razloga.

5. ZAKLJUČAK

Testiranje je veoma važan deo razvoja jednog softvera. Iako svi teže ka tome da pišu kod bez grešaka, one su neizbežne. Pogotovo su problematične one greške koje se možda i ne primete pri samom razvoju softvera, već kada on završi kod klijenta. Takve greške se neretko jako teško otklanjaju i potrebni su sati ili dani da bi se otkrio uzrok istih. Takođe, napretkom tehnologije se aplikacije mnogo brže prave, kraći su rokovi za izradu, a to donesi i veću šansu za greške.

Ukoliko je reč o softveru koji je vezan za određene industrije, greške mogu imati fatalne posledice. Zbog svega navedenog testiranje je jako bitno. U ovom radu je korišćen Jasmine okvir za testiranje. Testiranje se izvršavalo nad Angular aplikacijom. Prikazani su primeri testova svih najvažnijih delova jedne Angular aplikacije kao što su komponente, servisi, pajpovi i moduli.

6. LITERATURA

[1] Evan Hahn, *JavaScript Testing with Jasmine*, 2013.

[2] Paulo Ragoha, *Jasmine JavaScript Testing*, 2018.

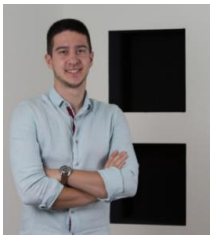
[3] Miodrag Živković, *Testiranje softvera*, 2018.

[4] https://www.innoq.com/en/blog/ts-jasmine-karma/

[5] https://testing-angular.com/

[6] https://medium.com/@turhan.oz/typescript-with-jasmine-easy-project-setup-530c7cc764e8

Kratka biografija:



Marko Vučković rođen je u Subotici 1995. godine. Završio je gimnaziju „Svetozar Marković“ u Subotici 2014. godine i upisao Fakultet Tehničkih Nauka u Novom Sadu iste godine. 2018 godine je završio osnovne akademske studije sa prosečnom ocenom 9.16.

kontakt:
markovuckovic1808@gmail.com