VAŽNOST TESTIRANJA ZA KVALITETNU IZRADU WEB RJEŠENJA

Radić, Kristian

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Algebra University College / Visoko učilište Algebra**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:225:568396

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

ALGEBRA VISOKO UČILIŠTE

Repository / Repozitorij:

Algebra Univerity College - Repository of Algebra Univerity College

zir.nsk.hr

dabar DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

VAŽNOST TESTIRANJA ZA KVALITETNU IZRADU WEB RJEŠENJA

Kristian Radić

Zagreb, siječanj 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, datum.

Predgovor

Studiranje na Visokom učilištu Algebra jedno je od onih iskustava koje obilježi cijeli život. Na početku mog studentskog puta, nisam bio siguran koliko pripadam ovom svijetu informacijskih tehnologija. Međutim, Visoko učilište Algebra mi je u ove 3 i pol godine pružilo neprocjenjiva znanja iz programiranja do kojih sam vjerojatno nikada ne bih došao, a što se nastavlja i na mom diplomskom studiju razvoja računalnih igara. Na putu do svoje prvostupničke diplome, najviše mi je pomogla obitelj, svojom financijskom i moralnom podrškom te se njima prvima zahvaljujem. Zatim, veliki trag za vrijeme studija, ostavio je profesor Daniel Bele koji svoju motivaciju i entuzijazam bez iznimke prenosi na sve studente i maksimalno se trudi oko svakog pojedinca. Na kraju, naravno, želim zahvaliti svom mentoru, profesoru Renatu Barišiću na njegovom stručnom vodstvu i sjajnom pristupu.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

Za svaki proizvod napravljen od strane čovjeka postoji vjerojatnost da u nekom trenutku zakaže i prestane obavljati funkciju zbog koje je stvoren. Softverska rješenja nisu u tome nikakva iznimka. Prestanak rada softverskih rješenja može uzrokovati korisnicima i proizvođačima tih rješenja veliku financijsku štetu, a u nekim slučajevima i uvelike narušiti ugled.

U ovom radu navedeni su neki od primjera važnosti ispravnog i detaljnog testiranja softvera, objašnjeni su principi za uspješno testiranje te su istražene vrste testiranja. Budući da je testiranje web rješenja jedno zaista opširno polje i kad bi se krenulo u razrađivanje svakog faktora uključenog u proces web testiranja, opseg materijala bi daleko nadmašio opseg završnog rada i prerastao u knjigu. Iz tog razloga se u ovom radu pokušalo obuhvatiti samo najbitnije vrste testiranja web rješenja, opisati njihove karakteristike, proces izvođenja, tijek rada, tipove i alate potrebne za automatizaciju njihovog izvođenja.

Za svrhu ovog rada, testirano je web rješenje BiGi PRO. Testiranje web rješenja je obavljeno prolazeći kroz svaki tip testiranja opisan unutar rada, po pravilima opisanim u tekstu. Alati koji su korišteni za testiranje su: LambdaTest, LoadView i OWASP ZAP.

For every product made by man, there is a probability that, at the certain moment, it will fail and stop performing the function it was made to do. Software solutions aren't an exception. Failure of software soultion can cause financial damage and sometimes greatly reduce reputation to users and developers of the solution.

In this thesis, some examples of importance of correct and detailed software testing are listed, principles of successful testing are explained and types of testing are studied. Since web testing is one truly wide field, if this was done by elaborating each factor included in the web testing process, the scope of materials would fairly surpass the scope of the final thesis and would have grown into a book. For that reason, in this thesis, I have tried to include only the most important types of web testing, describe their characteristics, process of performing them, workflow, subtypes and tools that are necessary for their automatization.

For purpose of this thesis, web solution BiGi PRO has been tested. Testing of the web solution has been done by conducting each type of testing describe in the thesis, by rules described inside the text. Tools used for testing are: LambdaTest, LoadView and OWASP ZAP.

Sadržaj

Predgovor 4

Sažetak 6

Uvod 1

1. Općenito o testiranju 2

1.1 Važnost testiranja 4

1.2 Principi uspješnog testiranja 6

2. Funkcionalno testiranje 8

2.1 Tipovi funkcionalnog testiranja 9

2.1.1 Testiranje jedinica *(engl. unit testing)* 9

2.1.2 Smoke testing 9

2.1.3 Testiranje ispravnosti *(engl. sanity testing)* 10

2.1.4 Regresijsko testiranje *(engl. regression test)* 10

2.1.5 Integracijsko testiranje *(engl. integration testing)* 11

2.1.6 Testiranje sustava *(engl. system testing)* 11

2.2 Funkcionalno testiranje u primjeni 11

3. Testiranje upotrebljivosti 15

3.1 Proces testiranja upotrebljivosti 16

3.2 Metode provedbe testiranja upotrebljivosti 17

3.3 Testiranje upotrebljivosti u primjeni 17

4. Testiranje sučelja 18

4.1 Faze testiranja sučelja 18

4.2 Testiranje sučelja u primjeni 19

5. Testiranje kompatibilnosti 22

5.1 Testiranje kompatibilnosti browsera 23

5.1.1 Tijek rada za testiranje kompatibilnosti browsera 24

5.1.1 Alati za testiranje kompatibilnosti browsera 26
5.1.2 Testiranje kompatibilnosti browsera u primjeni 27
6. Testiranje performansi 30
6.1 Koraci testiranja performansi 31
6.2 Testiranje opterećenja 31
6.2.1 Proces testiranja opterećenja 32
6.3 Testiranje otpornosti na stres 33
6.3.1 Tipovi testiranja otpornosti na stres 34
6.3.2 Proces testiranja otpornosti na stres 35
6.4 Alati za testiranje performansi 36
6.5 Testiranje performansi u primjeni 36
7. Sigurnosno testiranje 40
7.1 Tehnike testiranja sigurnosti 42
7.3 Alati za testiranje sigurnosti 44
7.4 Sigurnosno testiranje u primjeni 45
8. Zaključak 47
Popis kratica 48
Popis slika 49
Popis tablica 50
Literatura 51

Uvod

Bugovi, „softverski problemi koji uzrokuju neispravan rad rješenja, a posljedica su pogrešne logike (https://www.techopedia.com/definition/24864/software-bug- 31.01.2020.)“, su svakodnevica svakog poduzeća koje se bavi informacijskim tehnologijama. Posljedice ne detektiranih bugova su brojne, a sežu od nezadovoljstva korisnika, preko vremena koje se izgubi kako bi se bug naknadno popravio, pa sve do značajnih novčanih gubitaka koji se pojavljuju zbog dodatnog rada i utroška resursa koje poduzeća ulažu kako bi popravili svoje programsko rješenje nakon što je već došlo u doticaj s korisnicima. Prema istraživanju iz 2013. godine koje je provelo sveučilište na Cambridgeu, greške u softveru godišnje koštaju globalnu ekonomiju 312 milijuna dolara (http://insight.jbs.cam.ac.uk/2013/financial-content-cambridge-university-study-states-software-bugs-cost-economy-312-billion-per-year/ 23.10.2019.)

Testiranje programskih rješenja je faza razvoja programskog rješenja koja se u stručnim krugovima cijeni manje nego neke druge faze, iako nije ništa manje važna od ostalih. Ovaj rad će nastojati što bolje objasniti radne procese kroz koje je potrebno proći kako bi testiranje bilo uspješno obavljeno. Prikazati će se programska rješenja koja se pri tom koriste, kao i način rada u istima, a sve s ciljem naglaska na važnost testiranja.

1. Općenito o testiranju

„Testiranje programskih rješenja je proces određivanja ispravnosti rješenja kroz procjenu svih njegovih atributa (pouzdanost, skalabilnost, portabilnost, iskoristivost, mogućnost ponovnog korištenja) i izvođenje koda sa svrhom pronalaska bugova, grešaka ili nedostataka. (https://www.javatpoint.com/software-testing-tutorial 12.10.2019.)“ Obuhvaća testiranje svih komponenti potrebnih kako bi se smatralo da rješenje ispunjava sve kriterije koje ga čine zadovoljavajućim. Testiranje je jako bitna stavka u razvoju programskog rješenja jer, ako u bilo kojem trenutku to rješenje zakaže, to može biti potencijalno prilično opasna situacija za klijente i developere koji to rješenje razvijaju. Dakle, bez kvalitetno obavljenog testiranja, rješenje se ne smije dati klijentima na korištenje.

„Testiranje obuhvaća grupu tehnika kojima se određuje ispravnost rješenja prema predefiniranoj skripti, ali ne pronalazi sve nedostatke unutar rješenja. Glavna namjera mu je detektirati greške kako bi se mogle ispraviti. Bitno je naglasiti kako se kroz testiranje ne osigurava ispravan rad rješenja u svim uvjetima, već samo u specifičnim uvjetima pod kojima je određeno da mora raditi (https://www.javatpoint.com/software-testing-tutorial 12.10.2019.)“.

„Osobe koje obavljaju testiranje zovu se testeri. U prošlosti, u poduzećima koja se bave izradom programskih rješenja, nije bilo neobično da funkciju programera obavlja osoba koja ne zna testirati, a funkciju testera osoba koja ne zna programirati. Razlog tome je to što je testiranje povijesno nije bilo smatrano tehničkom aktivnošću, već se primarno gledalo s upravljačke i procesne strane. Međutim, kako je IT industrija rasla, tako su i programska rješenja postala sve više integrirana u svakodnevne živote ljudi, a s time je na važnosti dobila pouzdanost softvera, njegova održivost i njegova sigurnost. Kao posljedica toga, pojavila se potreba za poboljšanjem načina testiranja programskih rješenja, pa je tako danas testerima potrebna veća razina tehničkih znanja, kao što je i programerima došlo do veće potrebe za testerskim znanjima (Paul Ammann; Jeff Offutt; Introduction to software testing (2008.))“.

Što se tiče testiranja web rješenja, na koje će se ovaj rad fokusirati, „to je skup praksi koje testeri primjenjuju za testiranje kompletnog web rješenje prije nego što se ono da na korištenje korisnicima. (https://www.softwaretestinghelp.com/web-application-testing/ 12.10.2019.)“ Web testiranje osigurava da web rješenje funkcionira ispravno, te da se korisnici mogu služiti

njime u stvarnom vremenu. Kako bi se web rješenje ispravno testiralo, prema članku https://usersnap.com/blog/web-application-testing/ (12.10.2019.) potrebno je obaviti sljedećih 6 vrsta testiranja:

- **Funkcionalno testiranje** - testiranje s ciljem dokazivanja kvalitete softvera i vrsta black-box testa (testiranje bez poznavanja koda rješenja). Testiraju se sve funkcije rješenja davanjem nekog inputa i analiziranjem outputa.
- **Testiranje iskoristivosti** - kombinira funkcionalno testiranje i korisničko iskustvo, a mogu ga obaviti interni i eksterni testeri. U ovoj vrsti testiranja, testira se lakoća navigacije, jednostavnost uporabe, subjektivno zadovoljstvo korisnika i općenito izgled rješenja.
- **Testiranje sučelja** - testiranje sučelja osigurava da sve interakcije između web poslužitelja i rješenja teku glatko i bez smetnji. Uključuje komunikacijske procese i prikaz grešaka. Testira i prekide (*engl. interruptions)* od strane klijenta i od strane poslužitelja.
- **Testovi kompatibilnosti** - potrebno je osigurati da rješenje radi na različitim preglednicima (*engl. browsers)* i uređajima. To se osigurava kroz:
  - Test kompatibilnosti preglednika;
  - Test kompatibilnosti operacijskog sustava;
  - Test kompatibilnosti uređaja.
- **Testiranje performansi** - često zanemareno, obavlja se nakon osiguravanja da rješenje radi na svim preglednicima i uređajima. Na rješenje se stavlja teško opterećenje. Podrazumijeva testiranje rješenja na različitim brzinama interneta i testiranje njegovog ponašanja pod normalnim i vršnim opterećenjem (*engl. load testing*). Također, testira se koja je granica „pucanja“ rješenja tako što se na njega stavlja sve veći pritisak (*engl. stress testing*), sve dok ne prestane raditi.
- **Testiranje sigurnosti** - Posljednji korak testiranja web rješenja je provjera da li je rješenje zaštićeno od neautoriziranog pristupa štetnih akcija putem virusa i sličnih zlonamjernih programa. Testiranje sigurnosti obuhvaća provjeru mogućnosti pristupa sigurnim stranicama bez autorizacije, provjeru zatvaranja sesija (*engl. sessiona*) nakon neaktivnosti korisnika, verifikaciju SSL-a, osiguravanje da se zabranjene datoteke ne mogu preuzeti bez autorizacije. Kod testiranja sigurnosti potrebno je osigurati: Siguran prijenos podataka, autentifikaciju, upravljanje sesijom, autorizaciju, kriptografiju,

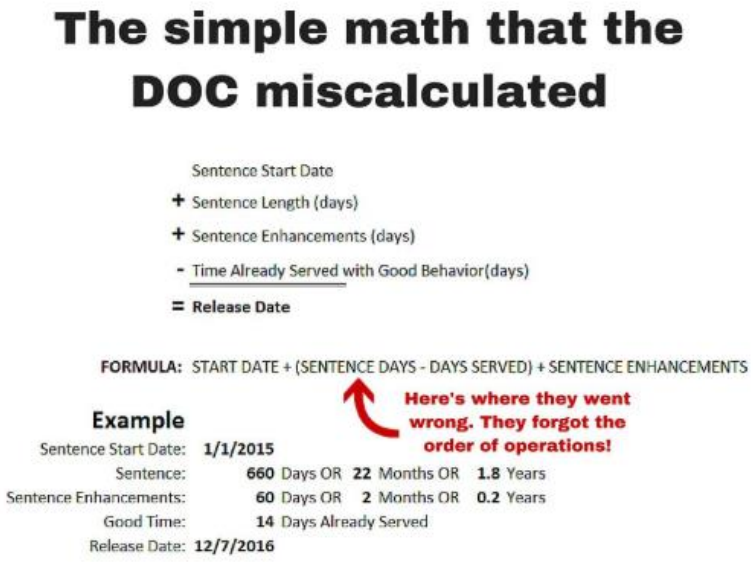
validaciju podataka, osigurati se od DOS napada, Testirati specifične funkcionalnosti i osigurati rješenje pri pojavi grešaka.

1.1 Važnost testiranja

U današnje vrijeme, kad se svijet ubrzano mijenja, Internet se u potpunosti integrirao u ljudske živote i mnogo ljudi često donosi odluke uz pomoć internetske pretrage. Uzevši to u obzir, da se zaključiti da web rješenja više nisu opcionalni, već obavezni dio svakog poslovanja, te su prvi korak u probijanju i ostajanju na tržištu. Web testiranje je praksa koja za cilj ima otkrivanje bugova unutar web rješenja, te samim time smanjenje troška naknadnog popravka bugova. Popravljanje bugova košta sve više što se bug kasnije otkrije. Prema IBM-u (https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/ 12.10.2019.), trošak popravljanja buga nakon puštanja rješenja u korištenje je 4 do 5 puta veći nego kad se bug otkrije za vrijeme dizajna i čak do 100 puta veći od popravljanja buga otkrivenog u fazi održavanja. Istraživanje koje je 2013. godine provelo sveučilište na Cambridgeu, otkriva da bugovi godišnje koštaju globalnu ekonomiju 312 milijardi dolara. S ekonomske strane, te brojke su dokaz važnosti testiranja programskih rješenja. Izvršavanjem testiranja prije nego što se rješenje da korisnicima na korištenje, osigurava se dobar rad rješenja i moguće je uštedjeti velik novac. Za primjer možemo uzeti otvaranje Terminala 5 na londonskom aerodromu Heathrow 2008. godine kad je došlo do greške u sustavu za upravljanje prtljagom (http://www.poslovni.hr/strane-kompanije/novi-terminal-londonskog-heathrowa-aerodromu-donio-dodatne-probleme-75596 21.11.2019.). Incident je prouzrokovao velike gužve i otkazivanje letova, a velik broj putnika je bio prisiljen prespavati na podu novog terminala. Uz sve to, ekonomska šteta tog dana se procjenjuje na 25 do 50 milijuna funti.

Osim financijske uštede, kvalitetno i temeljito testiranje programskog rješenja može korisnicima uštedjeti vrijeme, poštedjeti ih neugode, a neizostavna je i sigurnosna važnost testiranja. Programska rješenja su danas prisutna u gotovo svim uređajima oko nas, od mikrovalne pećnice i televizora, preko mobilnih uređaja i računala, pa sve do automobila, zrakoplova i svemirskih letjelica. Zbog takve sveprisutnosti programskih rješenja, potrebno je osigurati zadovoljstvo njihovih korisnika kroz pouzdanost i ispravnost rada. Naravno, ako se greška dogodi pri radu mikrovalne pećnice (oštećeni su korisnik i proizvođač mikrovalne pećnice), šteta je puno manja nego kad se ista dogodi, na primjer, u radu zrakoplova (prije

svega, ugroženi ljudski životi, a uz to i velika financijska šteta). Tako su se u listopadu 2018. godine dogodile dvije nesreće pri kojima se srušio novi model zrakoplova Boeing 737 Max (https://www.bloomberg.com/news/articles/2019-07-27/latest-737-max-fault-that-alarmed-test-pilots-rooted-in-software 21.11.2019.). Naknadno je otkriveno da je za te nesreće zaslužna bila greška u kodu koja je pri izravnavanju zrakoplova pomicala prednji dio zrakoplova previše prema dolje, ne ostavljajući pilotima dovoljno vremena za reakciju čime su bili prouzročeni padovi. Još jedan primjer koji ukazuje na važnost ispravnosti programskih rješenja je greška koja se dogodila u washingtonskom sustavu za kontrolu zatvorskih kazni (https://www.injurytriallawyer.com/blog/where-did-the-department-of-corrections-go-wrong-.cfm 21.11.2019.). Naime, zbog krivog izračuna, u periodu između 2002. i 2015. godine, iz washingtonskih zatvora je prerano oslobođeno oko 3.200 zatvorenika (Slika 1).



Slika 1: Greška zatvorskog sustava

Ovi primjeri pokazuju koliko testiranje programskih rješenja može biti delikatan posao i koliko je važno da se obavi ispravno. Iako je u ljudskoj prirodi da se greške uvijek događaju i nije moguće uvijek sve obuhvatiti, a posljedice se razlikuju od slučaja do slučaja, testeri uvijek moraju imati na umu da svaka greška u kodu koju ne uspiju pronaći ostavlja posljedice na korisnika.

„Usprkos svemu, u svijetu se još uvijek ne radi dovoljno na obrazovanju novih testera. Dokaz tome je rijetkost kolegija koji se bave testiranjem na preddiplomskim studijima fakulteta, a gotovo nijedan diplomski studij ne zahtjeva položen kolegij iz testiranja, dok istovremeno postoji tek nekolicina izbornih kolegija vezanih uz testiranje. Većina studenata informatike i računarstva nikad ne stekne nikakvo znanje iz testiranja, ili odslušaju samo nekoliko predavanja

koji su predviđeni kao dio kolegija o programskom inženjerstvu“ (Paul Ammann; Jeff Offutt; Introduction to software testing (2008.)).

1.2 Principi uspješnog testiranja

Kako bi se postigli optimalni rezultati testiranja, prema članku https://www.guru99.com/software-testing-seven-principles.html (22.11.2019.) preporučljivo je držati se određenih principa. Pri provođenju testiranja, potrebno je uzeti u obzir sve scenarije prema kojima se programsko rješenje može testirati. Ukoliko se svi scenariji uspiju obuhvatiti, vrijeme izvođenja rješenja će značajno porasti. Pri tome pomaže ako se testeri drže sljedećih 7 principa:

- **Iscrpno testiranje nije moguće** – naravno, pri provođenju testiranja nije moguće testirati apsolutno sve slučajeve u kojima programsko rješenje može biti pokrenuto. Razlog tome je velik izbor kombinacija uređaja, operacijskih sustava i preglednika koji pokreću rješenje. Također, što se testiranje detaljnije provodi, to dulje traje, a samim time i košta više, tako da neke stvari jednostavno nije isplativo testirati. Zbog toga je potrebno odrediti prioritete te dobro procijeniti do koje granice je isplativo testirati rješenje. Naravno, naglasak je potrebno staviti na kritične funkcije koje mogu uzrokovati rušenje rješenja. Jednako tako, nije potrebno testirati rješenje za uređaje i preglednike koji se ne koriste često (ne uzrokuju veliki gubitak u slučaju da nešto nije u redu).
- **Prikupljanje mana** – unutar ovog principa je vidljiva primjena Paretovog zakona na testiranje softvera. To znači da se otprilike 80% problema nalazi unutar 20% modula. Ovo pravilo, naravno, ne znači da se ostalih 80% modula može zanemariti, već samo ukazuje na što je potrebno obratiti više pažnje.
- **Paradoks pesticida** - naziv ovog principa proizlazi iz usporedbe ponavljajuće uporabe istih testova na softveru s ponavljajućom uporabom istih pesticida pri uzgoju voća i povrća. Korištenje istih pesticida s vremenom postane neučinkovito jer kukci razviju otpornost na njih. Isto tako, korištenje istih testova postaje neučinkovito jer na taj način nije moguće otkriti nove mane. Kako bi se spriječila neučinkovitost testova, potrebno je nad njima redovito vršiti reviziju te ih usavršavati i dodavati im nove, drugačije testove kako bi se pronašlo još grešaka i nedostataka. Testeri se ne mogu oslanjati na postojeće tehnike testiranja koje koriste, već konstantno moraju tražiti načine da ih

poboljšaju kako bi povećali učinkovitost. Unatoč tome, nikada nije moguće sa sigurnošću reći da određeni proizvod nema mana.

- **Testiranje ukazuje na postojanje mana –** ovaj princip nam govori da se testiranje fokusira na ukazivanje postojanja mana, ali ne i na odsutnost istih. Dakle, iako testiranje smanjuje vjerojatnost postojanja neotkrivenih grešaka i mana u softveru, ono nije dokaz ispravnosti rada, pa čak i onda kad nikakve greške i mane nisu pronađene.
- **Nedostatak grešaka može dovesti do zablude –** u slučaju da je programsko rješenje 99% bez greške, još uvijek može biti neiskoristivo. Ovo se događa kad se rješenje testira temeljito, ali za pogrešan zahtjev. Testiranje softvera nije samo traženje grešaka. Ono također mora provjeriti da programsko rješenje ispunjava poslovne potrebe korisnika. To što rješenje radi savršeno ne znači puno u slučaju da ono ne donosi korist korisniku.
- **Rano testiranje –** testiranje bi trebalo početi što ranije moguće unutar SDLC-a *(engl. Software Development Life Cycle).* Kao što je navedeno ranije u radu, što se ranije otkrije da nešto nije u redu, jeftiniji je popravak tog problema. Preporučeno je potragu za greškom započeti onog trenutka kad se definiraju zahtjevi koje programsko rješenje treba ispuniti.
- **Testiranje ovisi o kontekstu –** različita programska rješenja se neće testirati na jednak način. Video igra, web trgovina i poslovna desktop aplikacija će biti testirane na tri potpuno različita načina. Postoje različiti pristupi, metodologije, tehnike i tipovi testiranja koji se razlikuju u odnosu na tip programskog rješenja. Na testeru je da procijeni što je prikladno za rješenje koje mora testirati.

2. Funkcionalno testiranje

„Funkcionalno testiranje je vrsta black box testiranja koja se provodi kako bi se utvrdila funkcionalnost programskog rješenja i ponaša li se rješenje onako kako je očekivano. (https://www.softwaretestinghelp.com/guide-to-functional-testing/ 24.11.2019.)“ To znači da testeri ne smiju znati logiku unutar rješenja prilikom testiranja. Pri izvođenju funkcionalnog testiranja, testeri funkcijama daju određene inpute, a zatim analiziraju outpute. Ova vrsta testiranja se ne fokusira na način kojim se određeni proces izvodi unutar rješenja, već isključivo daje li taj proces ispravne rezultate. Zbog toga se funkcionalno testiranje može smatrati i simulacijom korištenja rješenja u stvarnom životu. Prema članku http://softwaretestingfundamentals.com/functional-testing/ (24.11.2019.), uobičajeni koraci koji za izvođenja funkcionalnog testiranja su:

- Identifikacija funkcija koje programsko rješenje treba izvesti;
- Stvaranje podataka za input na temelju specifikacija funkcije;
- Određivanje outputa na temelju specifikacija funkcije;
- Izvršavanje testnog slučaja;
- Usporedba očekivanih rezultata s dobivenim.

Navedeni koraci su učinkovitiji ukoliko su provedeni na temelju korisničkih zahtjeva. Kad se testovi provode na temelju dokumentacije rješenja, postoji rizik od previda određenih mana sustava, što može uzrokovati nezadovoljstvom korisnika.

Naravno, ova vrsta testiranja ne bi funkcionirala kad se ne bi znalo kakvo ponašanje rješenja je prihvatljivo, a kakvo nije. To se određuje na temelju specifikacije zahtjeva, ali također je potrebno uzeti u obzir i poslovne scenarije. S obzirom na to, prema članku https://www.softwaretestinghelp.com/guide-to-functional-testing/ (24.11.2019.), funkcionalno testiranje se može izvesti koristeći dvije popularne tehnike:

- **Testiranje utemeljeno na zahtjevima** – sadrži sve funkcionalne specifikacije koje tvore osnovu svih testova koji će se provesti.
- **Testiranje utemeljeno na poslovnim scenarijima** – sadrži informacije o tome kako će programsko rješenje biti percipirano iz perspektive poslovnih procesa.

2.1 Tipovi funkcionalnog testiranja

Postoji više tipova funkcionalnog testiranja. Svaki od njih može biti korišten, ovisno o prilici i scenariju testiranja koji tester želi provesti.

2.1.1 Testiranje jedinica *(engl. unit testing)*

„Testiranje jedinica je vrsta funkcionalnog testiranja pri kojem se testiraju samo određene komponente programskog rješenja. Svrha toga je određivanje funkcionira li svaka jedinica programskog rješenja onako kako je i dizajnirana. Jedinice su najmanji dijelovi programskog rješenja koji se mogu testirati. One obično imaju nekoliko inputa i jedan output. (http://softwaretestingfundamentals.com/unit-testing/ 21.11.2019.)“ Ovaj tip testiranja uglavnom izvode programeri, a u rjeđim slučajevima testeri. Neke od koristi ovog tipa testiranja su brže ispravljanje grešaka, čini kod prikladnijim za ponovno korištenje, čini proces razvoja programskog rješenja bržim zbog skraćivanja vremena potrebnog za pronalazak i ispravljanje grešaka, smanjuje cijenu popravka greške u odnosu na slučaj u kojem se greška pronađe na višoj razini i kod postaje pouzdaniji. Dokaz ovih koristi može biti sljedeći primjer. Recimo da tester na raspolaganju ima programsko rješenje koje se sastoji od dvije jedinice i testira ga tek nakon što je rješenje gotovo (preskače testiranje jedinica i integracijske testove). Pri traženju greške, već u ovako malom primjeru, gubi se puno vremena na određivanje je li greška u prvoj, drugoj ili možda čak u obje jedinice, je li greška u sučelju ili u samom testu.

2.1.2 Smoke testing

Ovaj tip vrsta testiranja provodi se nakon puštanja svakog *builda* na testiranje sa svrhom određivanja stabilnosti *builda*. „Obuhvaća testove većine glavnih funkcija programskog rješenja, ali samo površno. (http://softwaretestingfundamentals.com/smoke-testing/ 24.11.2019.)“ Prema rezultatima ove vrste testiranja određuje se da li je *build* dovoljno stabilan da bi se moglo nastaviti s njegovim daljnjim testiranjem. „Naziv „smoke testing“, navodno potječe od sličnog tipa testiranja hardvera u kojem uređaj prolazi test ukoliko se ne dimi (ili ne zapali) prilikom prvog pokretanja. (http://softwaretestingfundamentals.com/smoke-testing/ 24.11.2019.)“ Ukoliko *build* prođe smoke test, šalje se na daljnje testiranje, ali ako ne prođe, potrebni su mu popravci. Odgovor na pitanje zašto se ovako površno testiranje uopće obavlja leži u uštedi vremena i truda koja se dobiva kad se po završetku ovakvog testiranja utvrdi da rješenje ima velikih problema. Smoke testovi mogu se provoditi ručno, ali i uz pomoć alata ili

skripti za automatizaciju. U slučajevima češćih *buildova*, najbolje je automatizirati smoke testiranje. Također, kako se opseg programskog rješenja povećava dodavanjem novih funkcionalnosti, tako testovi moraju biti više ekspanzivni jer je samo jedan krivi znak unutar koda dovoljan da se cijelo rješenje proglasi beskorisnim.

2.1.3 Testiranje ispravnosti *(engl. sanity testing)*

Ovaj tip testiranja se uglavnom radi nakon smoke testiranja. „Koristi se kako bi se osiguralo da su sve promjene koje su napravljene na kodu ispravne. (https://www.geeksforgeeks.org/sanity-testing-software-testing/ 2.12.2019.)“ Sličan je smoke testu po tome što se fokusira na glavne funkcionalnosti rješenja i ne zalazi previše u detalje. Također, poput smoke testa, služi kako bismo uštedjeli vrijeme i novac ukoliko rješenje ne prođe ovu vrstu testiranja. Testiranje ispravnosti obično nije skriptirano, niti dokumentirano. Neke od prednosti su mu to što pomaže u brzoj identifikaciji grešaka unutar glavne funkcionalnosti i provodi se unutar kratkog vremena jer ne zahtijeva dokumentaciju.

2.1.4 Regresijsko testiranje *(engl. regression test)*

„Regresijsko testiranje je definirano kao tip testiranja programskog rješenja kojemu je cilj potvrditi da promjene na kodu nisu negativno utjecale na postojeće funkcionalnosti. Regresijsko testiranje je zapravo ponavljanje već provedenih testova kako bi se osiguralo da sve radi kako treba nakon promjena. Ovaj tip testiranja je potrebno izvršavati kad se dogodi promjena zahtjeva i kod se promijeni sukladno sa zahtjevom, kad se dodaje nova funkcionalnost, kad se popravlja greška i kad se popravljaju problemi s performansama. (https://www.guru99.com/regression-testing.html 2.12.2019.)“ Regresijsko testiranje se, prema prethodno referenciranom članku, može provesti kroz nekoliko tehnika:

- **Ponovno izvođenje svih testova** – ova tehnika je jako skupa, jer zahtijeva puno vremena i resursa.
- **Selekcija regresijskog testiranja** – umjesto ponavljanja svih testova, testni slučajevi se kategoriziraju u ponovljive testne slučajeve i zastarjele testne slučajeve.
- **Prioritizacija testnih slučajeva** – određuje se koje je testne slučajeve bitno provesti.

2.1.5 Integracijsko testiranje *(engl. integration testing)*

„Integracijsko testiranje je tip testiranja u kojem se kombiniraju jedinice programskog rješenja i potom se testiraju kao grupa. Svrha ovog tipa testiranja je pronaći greška u interakciji između integriranih jedinica. (http://softwaretestingfundamentals.com/integration-testing/ 2.12.2019.)“ Integracijsko testiranje provodi se nakon testiranja jedinica, a prema prethodno referenciranom članku, nekoliko je mogućih pristupa integracijskom testiranju:

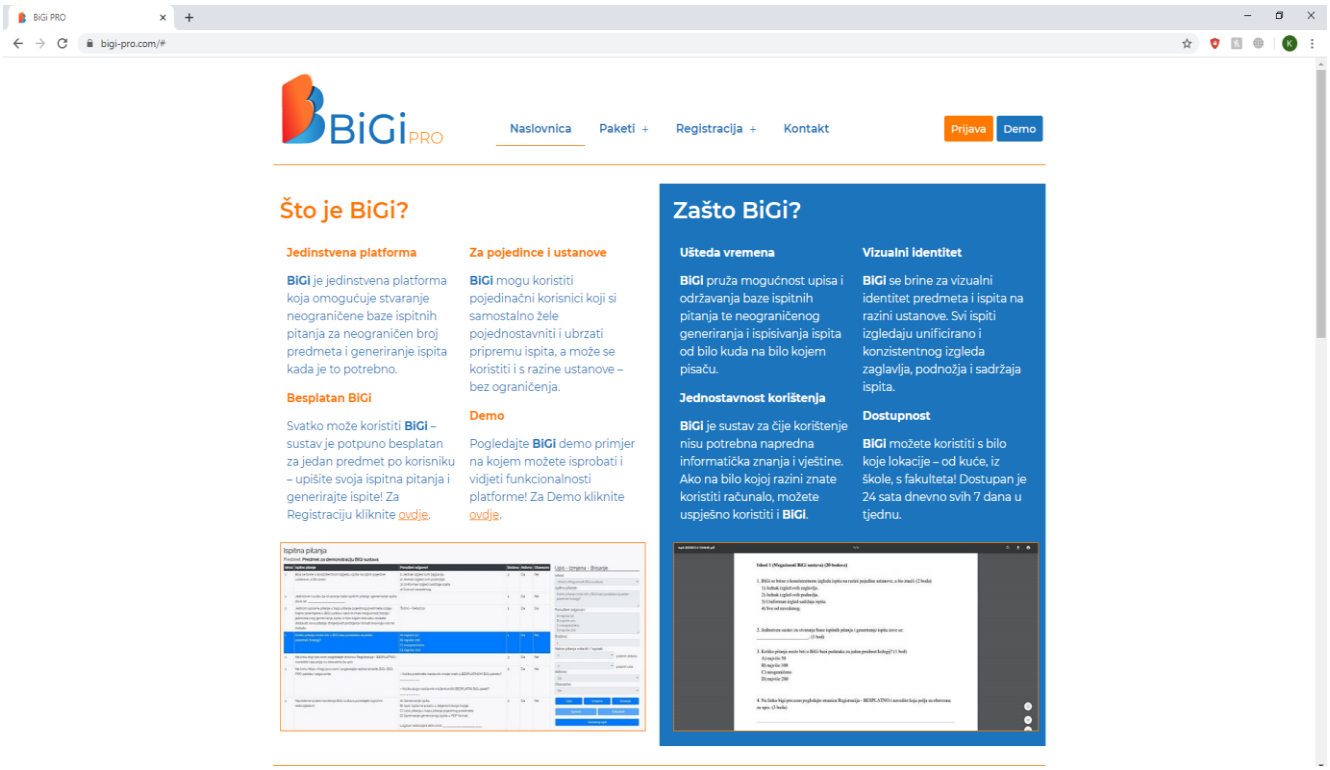
- **Big bang** – sve ili većina jedinica se kombiniraju i testiraju odjednom. Ovaj pristup se primjenjuje kad testeri dobiju kompletno programsko rješenje u paketu. Razlika između ovakvog načina integracijskog testiranja i testiranja sustava je u tome što se u ovom slučaju testiraju samo interakcije između jedinica.
- **Odozgo prema dolje *(engl. top down)*** – Jedinice viših razina se testiraju prve, a potom jedinice nižih razina. Ovaj pristup se koristi kad se primjenjuje i način razvoja programskog rješenja odozgo prema dolje.
- **Odozdo prema gore (*engl. bottom up*)** – Jedinice nižih razina se testiraju prve, a potom jedinice viših razina. Ovaj pristup se koristi kad se primjenjuje i način razvoja programskog rješenja odozdo prema gore.
- **Sendvič/Hibrid** – Kombinira pristupe odozgo prema dolje i odozdo prema gore.

2.1.6 Testiranje sustava *(engl. system testing)*

„Testiranje sustava je tip testiranja pri kojem se testira cjelokupno, integrirano programsko rješenje. Svrha ovog tipa testiranja je procjena usklađenosti rada programskog rješenja sa zahtjevima koje rješenje treba ispuniti. Provodi se nakon što su sve komponente programskog rješenja integrirane. (http://softwaretestingfundamentals.com/system-testing/ 2.12.2019.)“

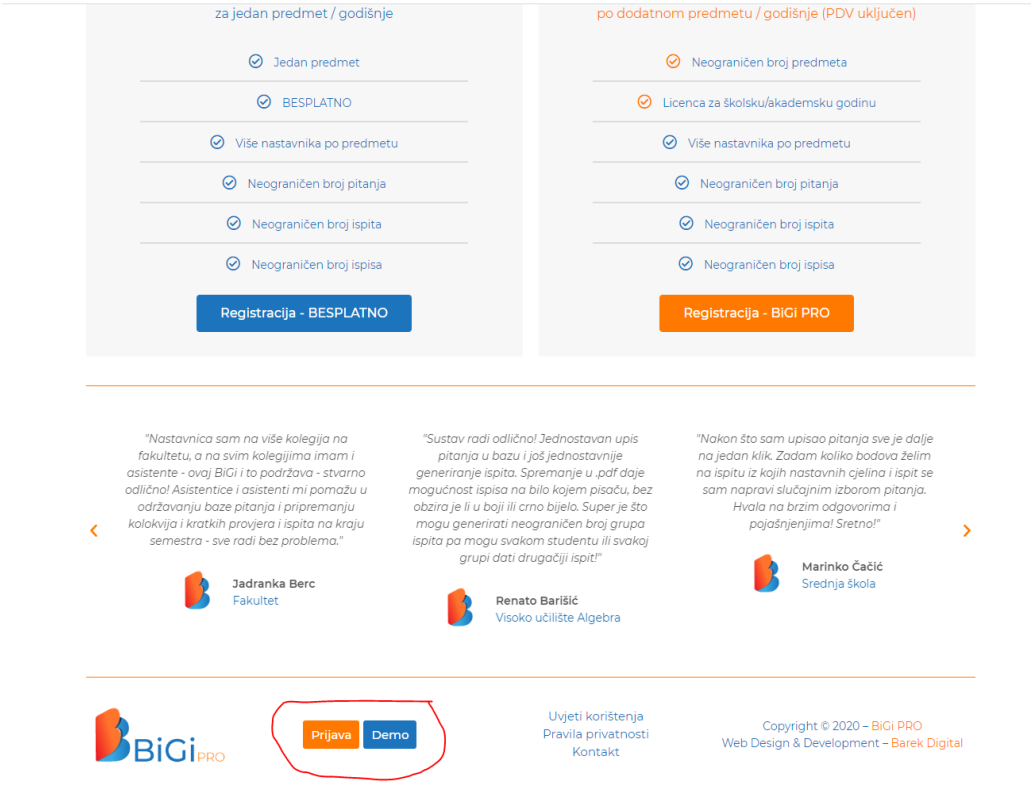
2.2 Funkcionalno testiranje u primjeni

U svrhu ovog rada, s obzirom na to da je testiranje izvođeno na već gotovom web rješenju BiGi PRO, obavljeno je testiranje sustava. U daljnjem tekstu će se prikazana testiranja također odnositi na web rješenje BiGi PRO. Identificirane su funkcije koje rješenje treba izvesti, stvoreni su podatci koje je potrebno unijeti unutar rješenja, izvršen je testni slučaj, te su uspoređeni dobiveni rezultati s onima koji su bili očekivani. Slika 2 prikazuje naslovnu stranicu testiranog rješenja.



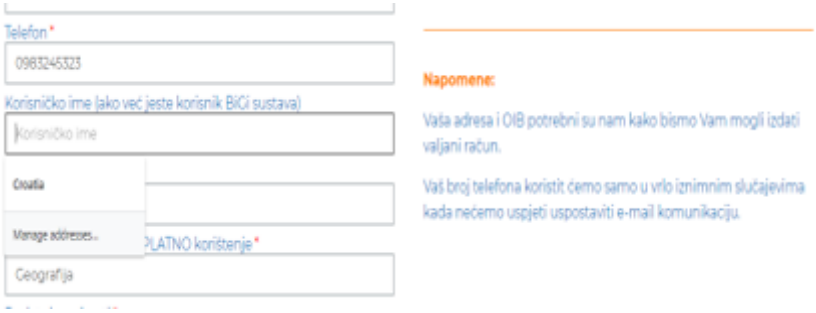
Slika 2: Naslovna stranica rješenja BiGi PRO

Provođenjem funkcionalnog testiranja utvrđeno je da gumbi „*Prijava*“ i „*Demo*“ ne vode na stranice koje su predviđene za prijavu i demo verziju. Ukoliko se navedene gumbe klikne na zaglavlju stranice, ne događa se ništa, a ukoliko ih se klikne u podnožju (Slika 3), vraćaju korisnika na vrh stranice.

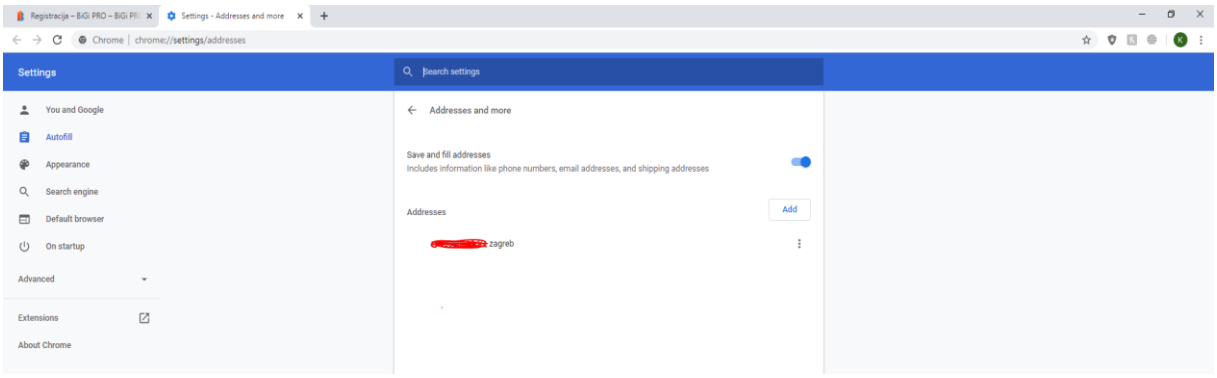


Slika 3: Prikaz podnožja BiGi PRO s gumbima "Prijava" i "Demo"

Nadalje, na formi za plaćenu registraciju, tekstni okvir predviđen za unošenje korisničkog imena nudi naziv države te opciju „*Manage adresses*“ (Slika 4), nakon čijeg odabira se otvara stranica browsera sa postavkama za upravljanje adresama (Slika 5).



Slika 4: Greška na tekstnom okviru - registracijska forma



Slika 5: Stranica za upravljanje adresama unutar browsera

Također, unutar iste forme, pod tekstnim okvirom koji je predviđen za OIB je moguće poslati neispravne podatke u smislu neispravnog broja znakova. OIB mora sadržavati točno 11 znakova, a moguće je poslati OIB bilo koje duljine.

Osim navedenih grešaka, ostale funkcionalnosti rješenja rade ispravno.

3. Testiranje upotrebljivosti

„Testiranje upotrebljivosti je način testiranja pri kojem se programsko rješenje testira od strane krajnjih korisnika, na način da korisnici koriste rješenje kako bi otkrili mane istog. (https://www.guru99.com/usability-testing-tutorial.html 10.12.2019)“ Na ovaj način se može lakše procijeniti jednostavnost uporabe programskog rješenja. Provodi se na način da se korisnicima daju određeni zadatci koje trebaju napraviti unutar rješenja pod nadzorom ispitivača čiji je zadatak primijetiti koji dijelovi programskog rješenja kod korisnika izazivaju zbunjenost i zadaju im probleme. Ukoliko više korisnika zapinje na istom mjestu, daju se preporuke za izmjenu. Budući da je proizvod iz oka korisnika uglavnom dobar onoliko koliko dobro izgleda, da se zaključiti da su estetika i dizajn bitni kod izrade programskog rješenja. Zbog toga testiranje upotrebljivosti daje odgovor na pitanja kao što su: „Gdje treba kliknuti?“, „Na koju stranicu treba navigirati?“, „Koja ikona što predstavlja?“, „Jesu li poruke o greškama konzistentne i efektivno prikazane?“ i „Traje li sesija dovoljno?“. Testiranje upotrebljivosti u ranijim fazama SDLC-a mogu spasiti programsko rješenje od neuspjeha.

Postoji nekoliko načina provedbe testiranja iskoristivosti prema članku https://www.experienceux.co.uk/faqs/what-is-usability-testing/ (10.12.2019.):

- **Komparativni test upotrebljivosti** – programsko rješenje se uspoređuje s drugim, konkurentnim rješenjem. Također se mogu uspoređivati i dva dizajna istog rješenja kako bi se zaključilo koji pruža bolje korisničko iskustvo.
- **Istraživački test upotrebljivosti** – korisnici testiraju rješenje po različitim, realističnim scenarijima kako bi pronašli nedostatke i prikazali na koje greške u dizajnu se potrebno fokusirati.
- **Evaluacija iskoristivosti** – korisnici testiraju novu ili ažuriranu uslugu kako bi se osigurala intuitivnost i pozitivno korisničko iskustvo. Svrha evaluacije iskoristivosti je osigurati da se potencijalni problemi istaknu prije nego što se rješenje pusti na korištenje.

Neke od prednosti testiranja iskoristivosti su direktna povratna informacija od strane ciljane publike, rješavanje unutarnjih nesuglasica na način da se promatra korisnikova reakcija na različite opcije i isticanje potencijalnih problema prije nego što se programsko rješenje pusti u uporabu. Također, s poslovne strane, prednosti koje su vidljive po završetku projekta su povećanje vjerojatnosti uporabe i ponovne uporabe rješenja te minimalizacija rizika od neuspjeha.

S druge strane, nedostatci testiranja iskoristivosti su to što testiranje nije 100% reprezentativno u odnosu na stvarne scenarije te činjenica da je testiranje iskoristivosti uglavnom kvalitativno pa ne osigurava veliku količinu povratnih informacija.

3.1 Proces testiranja upotrebljivosti

Prema članku https://www.guru99.com/usability-testing-tutorial.html (10.12.2019.), proces testiranja upotrebljivosti sastoji se od 5 faza (Slika 6):

- **Planiranje** – u ovoj fazi se određuju ciljevi testiranja upotrebljivosti. Potrebno je odrediti ključne funkcionalnosti i ciljeve rješenja. Dodjeljuje se zadatci testerima koji prolaze kroz te ključne funkcionalnosti. U ovoj fazi, određuju se i metoda testiranja, broj i demografija testera i format izvješća.
- **Regrutiranje** – u ovoj fazi se regrutira željeni broj testera, ovisno o planu testiranja upotrebljivosti. Pronalazak testera koji odgovaraju opisu idealnog testera rješenja može potrajati. Broj potrebnih testera ovisi o kompleksnosti rješenja i ciljevima testiranja. Naravno, povećanje broja testera rezultira i većim troškovima te većom potrebom za planiranjem, menadžmentom sudionika i analizom podataka. Ukoliko se raspolaže malim budžetom, kao dobar broj testera se uzima 5, no ukoliko budžet nije ograničenje, najbolje se posavjetovati sa stručnjacima.
- **Testiranje** – faza u kojoj se obavljaju konkretna testiranja.
- **Analiza** podataka – temeljito se analiziraju podatci dobiveni iz testiranja kako bi se izvuklo kvalitetne zaključke i postupilo u skladu s njima, s ciljem poboljšanja iskoristivosti rješenja.
- **Izvješće** – rezultati testa iskoristivosti se dijele sa svim zainteresiranim dionicima. To može uključivati dizajnere, developere, klijente, direktore, itd.



Slika 6: Proces testiranja upotrebljivosti

3.2 Metode provedbe testiranja upotrebljivosti

Dvije su metode provedbe testiranja iskoristivosti:

- **Laboratorijsko testiranje iskoristivosti** – odvija se u laboratorijskoj sobi uz nadzor promatrača. Testerima se dodjeljuju zadatci koje moraju izvršiti, a promatrači nadziru ponašanje testera i prijavljuju rezultate testiranja u izvješću. Promatrač se tijekom testiranja ne upliće u postupke testera i ne razgovara s njima.
- **Testiranje iskoristivosti s udaljene lokacije** – promatrači i testeri su, u ovom slučaju, smješteni na udaljenim lokacijama. Testeri pristupaju testu sa svoje lokacije te njihov glas, aktivnost na ekranu i izraze lica snima automatizirani softver. Promatrači analiziraju ove podatke i prijavljuju rezultate testa.

3.3 Testiranje upotrebljivosti u primjeni

U svrhu ovog rada, provedeno je laboratorijsko testiranje na uzorku od 5 testera pod nadzorom autora rada koji je obnašao ulogu promatrača testiranja. Testiranje je bilo istraživačkog karaktera, što znači da su korisnici prolazili kroz realistične scenarije kako bi evaluirali rad rješenja.

Po završetku testiranja, povratna informacija od strane korisnika je bila u glavnom dobra. Korisnici su kroz rješenje mogli navigirati bez zapreka. Jedini problem im je predstavljao rad gumba „*Paketi+*“ i „*Registracija+*“ (Slika 7). Korisnicima ovi gumbi nisu bili intuitivni jer se nalaze između gumba „Naslovnica“ i „Kontakt“ koje korisnika preusmjeravaju na drugu stranicu. Iako problematični gumbi rade na princip *drop down liste*, bilo bi dobro korisniku omogućiti da i klikom na prvobitno prikazane gumbe dospije na zasebnu stranicu.



Slika 7: Prikaz gumba zaglavlja BiGi PRO

4. Testiranje sučelja

„Programska rješenja se sastoje od više komponenti. Te komponente su server, baza podataka itd. Sučelje je veza koja spaja te komponente i putem koje obavljamo određene radnje na istima, to jest, ono je softver koji sadrži naredbe i poruke putem kojih korisnik komunicira s programskim rješenjem. Iz perspektive računala, sučelje može biti API, web servis i slično. Komunikacija između komponenti programskog rješenja mogu uvelike utjecati na izvršavanje rješenja. Testiranje sučelja je vrsta testiranja koja provjerava ispravnost komunikacije između dva različita rješenja ili dvije komponente istog rješenja. Ono osigurava da krajnji korisnik koristi programsko rješenje bez problema, identificira dijelove rješenja koje krajnji korisnik obično koristi, provjerava sigurnosne zahtjeve za vrijeme komunikacije između komponenti te provjerava može li rješenje izdržati greške u mrežnoj komunikaciji između web servera i rješenja. (https://www.educba.com/interface-testing/ 10.12.2019.)“

Razna web rješenja, poput Flipkarta i Amazona se sastoje od web aplikacije, baze podataka i servera. Kako bi se podatci prenijeli od jedne komponente do druge, ova rješenja koriste sučelja kao što su API i web servisi koja omogućuju korisniku da se prijavi kroz korisničko sučelje, a njegovi podatci se spreme u bazi podataka. Naravno, postoje i slučajevi u kojima neautorizirani korisnici pokušavaju pristupiti, koristiti ili spremati podatke do kojih ne bi smjeli doći. Kako bi se takvi korisnici spriječili u njihovoj namjeri, potrebno je dobro izvesti testiranje sučelja.

Testiranje sučelja mora obuhvatiti testiranje sučelja između web servera i aplikacije i testiranje sučelja između aplikacije i baze podataka. Pri tom je potrebno provjeriti izvršava li se sve kako treba, jesu li greške ispravno uhvaćene ili vraćaju poruke greške za bilo koji upit od strane aplikacije i provjeriti što se dogodi kad se resetira veza s web serverom.

4.1 Faze testiranja sučelja

3 su faze testiranja sučelja prema članku https://www.educba.com/interface-testing/ (10.12.2019.):

- **Konfiguracija i razvoj** – nakon konfiguracije sučelja i inicijalizacije razvoja, potrebno je verificirati konfiguraciju u skladu sa zahtjevima.
- **Validacija** – po završetku razvoja, obavlja se validacija sučelja. Ova faza se može izvesti i kao dio testiranja jedinica.

- **Održavanje** – jednom kad je programsko rješenje gotovo i pušteno u korištenje, sučelje je potrebno nadzirati i pratiti njegove performanse. Moguće je da se s vremenom pojave novi problemi zbog izvršenih promjena ili zbog opadanja u performansama. Pri uvođenju promjena, potrebno je obaviti testove koji će osigurati da te promjene nisu uzrokovale pojavljivanje novih grešaka na sučelju. Nakon što je taj posao obavljen, potrebno je obaviti validaciju podataka i tijeka rada *(engl. workflow)*.

4.2 Testiranje sučelja u primjeni

Što se tiče veza unutar same aplikacije BiGi PRO, testirane su prethodno u sklopu funkcionalnog testiranja i greške prikazane u tom dijelu su primjenjive i na ovo poglavlje. Što se tiče komunikacije sa drugim uređajima, ona se odvija unutar formi za kontakt, te registraciju. Slika 8 i Slika 9 prikazuju ispravno popunjene forme kontakta i registracije.



Slika 8: Ispravno popunjena kontakt forma



Slika 9: Ispravno popunjena registracijska forma

Nakon ispravnog popunjavanja, podatci se šalju administratoru rješenja na e-mail kao što prikazuju Slika 10 i Slika 11.



Slika 10: Primljeni e-mail s podatcima kontakt forme



Ime: Kristian
Prezime: Radic
E-mail: mail@mail.com
Naziv ustanove: Algebra
Naziv predmeta za BESPLATNO korištenje: Geografija
Napomene: Test sucelja
Privola 1: on
Privola 2:

Date: February 15, 2020
Time: 8:45 am
Page URL: https://bigi-pro.com/registracija-besplatno/
User Agent: Mozilla/5.0 (Linux; Android 9; POCOPHONE F1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.99 Mobile Safari/537.36
Remote IP: 89.164.156.5

Slika 11: Primljeni e-mail s podatcima registracijske forme

5. Testiranje kompatibilnosti

„Kompatibilnost je sposobnost dvaju entiteta da funkcioniraju zajedno bez nedosljednosti. Ista definicija vrijedi i za programska rješenja. Kako bi se programsko rješenje smatralo kompatibilnim, ono mora raditi unutar svih okruženja za koja je predviđeno (https://www.softwaretestinghelp.com/software-compatibility-testing/ 12.12.2019.).“

„Testiranje kompatibilnosti je vrsta nefunkcionalnog testiranja. Ono je vrsta softverskog testiranja kojim se osigurava kompatibilnost programskog rješenja s okruženjem unutar kojeg se pokreće. Provjerava se izvršava li se programsko rješenje ispravno na različitim uređajima, browserima, operacijskim sustavima, itd. (https://www.softwaretestinghelp.com/software-compatibility-testing/ 12.12.2019.)“ Na rad programskog rješenja također mogu utjecati i različite verzije, rezolucija, brzina interneta, konfiguracija i slično. Testiranje kompatibilnosti je poželjno izvršavati u stvarnom okruženju, a ne u virtualnom, kako bi se smanjila mogućnost greške. Pri izvršavanju testiranja kompatibilnosti, u fokusu nisu rezultati koji se dobivaju prilikom unošenja određenih inputa, već samo način na koji rješenje radi u različitim okruženjima.

Kompatibilnost programskog rješenja s njegovim okruženjem je jako bitno za zadovoljstvo krajnjeg korisnika. Pri prodaji određenog programskog rješenja, nije moguće znati okruženje u kojem korisnik to programsko rješenje želi pokrenuti, a ukoliko, nakon kupnje rješenja, korisnik shvati da ono ne radi ili da ima mnogo grešaka, njegovo zadovoljstvo pada, a proizvođaču rješenja pada ugled. Zbog toga je bitno pri testiranju kompatibilnosti obuhvatiti što više različitih okruženja. Naravno, nije moguće pokriti sve browsere, operacijske sustave i uređaje, pa je zbog toga bitno napraviti procjenu isplativosti kako bi se znalo u kojim okruženjima se ne želi da se programsko rješenje izvršava.

Testirati se može:

1) **Kompatibilnost hardvera** – testira se kompatibilnost rješenja s raznim konfiguracijama hardvera kako bi se osigurala njegova ispravnost. U ovo spada i testiranje kompatibilnosti raznih operacijskih sustava (Mac OS X, Windows, Unix…).
2) **Kompatibilnost softvera** – testira se kompatibilnost softvera s različitim drugim softverima. Pod ovo spadaju:
   a) **Testiranje kompatibilnosti mreže** – testni tim provodi evaluaciju performansi rješenja na mreži s različitim parametrima kao što su propusnost *(engl. bandwith),* brzina, kapacitet itd. Također potvrđuje ispravnost rada rješenja na različitim mrežama.

b) **Testiranje kompatibilnosti browsera** – testni tim potvrđuje da je rješenje kompatibilno i ispravno radi s različitim browserima, kao što su IE, Google Chrome, Mozilla Firefox itd. Na ovu kategoriju testiranja kompatibilnosti ćemo se dodatno fokusirati dalje u radu.

c) **Testiranje kompatibilnosti uređaja** – provjerava se kompatibilnost rješenja s raznim mobilnim uređajima s različitim operacijskim sustavima i veličinama ekrana (iOS, Android).

d) **Testiranje verzija softvera** – provjerava se ispravnost rada softvera na različitim verzijama istog. Ova vrsta testiranja se prema članku https://www.testbytes.net/blog/compatibility-testing/ (12.12.2019.) može provesti na 2 načina:

- Testiranje kompatibilnosti unatrag – testni tim provjerava je li rješenje kompatibilno sa starijim verzijama platformi.
- Testiranje kompatibilnosti unaprijed – testni tim provjerava kompatibilnost rješenja s budućim verzijama platformi. Teže je za izvesti od testiranja unatrag jer članovima tima nisu uvijek poznato sve što je potrebno za izvesti ovu vrstu testiranja.

5.1 Testiranje kompatibilnosti browsera

„Tehnologija kroz godine napreduje velikom brzinom i pojavljuju se novi browseri i nove verzije starih browsera. Međutim, velik broj ljudi pruža otpor promjenama, pa tako postoji i velik broj korisnika koji koriste stare verzije browsera. Testiranje kompatibilnosti browsera je bitno upravo radi zadovoljavanja potreba tih korisnika. Iako se većina programskih rješenja razvija kako bi ih se pokretalo na najnovijim browserima radi što boljih performansi, broj ovih korisnika je prevelik da bi ih se ignoriralo. (https://www.browserstack.com/guide/cross-browser-compatibility-testing-beyond-chrome 12.12.2019.)“ Kad se programsko rješenje ne bi razvijalo na način da radi i na starijim verzijama browsera, velik dio tržišta bi ostao neiskorišten, što bi rezultiralo neiskorištavanjem punog tržišnog potencijala programskog rješenja, što znači i znatno manju zaradu. Budući da različiti browseri imaju različite konfiguracije, bez ove vrste testiranja, pri promjeni browsera je moguće pojavljivanje znakova nekompatibilnosti na različitim razinama. Nekompatibilnost se može očitovati u neispravnom izgledu, radu, pa čak i smanjenoj brzini izvršavanja funkcija programskog rješenja. Zbog toga je, prema članku https://developer.mozilla.org/en-

US/docs/Learn/Tools_and_testing/Cross_browser_testing/Introduction (13.01.2020.) pri testiranju kompatibilnosti browsera potrebno imati na umu:

- Starije browsere koji su još uvijek u uporabi, a koji ne podržavaju najnovije značajke CSS-a i JavaScripta;
- Različite uređaje na kojima se pokreću isti browseri, a koji zbog različitosti nemaju jednake mogućnosti (npr. tableti, smartphonei, pametni televizori, stariji mobilni uređaji…);
- Ljude sa invaliditetima koji koriste Internet uz pomoć uređaja poput čitača zaslona.

5.1.1 Tijek rada za testiranje kompatibilnosti browsera

Kako testiranje kompatibilnosti rješenja u različitim browserima ne bi oduzelo previše vremena, potrebno ga je pažljivo isplanirati i osigurati da se testiranje obavlja na pravim mjestima. Ukoliko se radi na velikom projektu, potrebno je testirati na redovnoj bazi kako bi se osiguralo da nove značajke rješenja funkcioniraju ispravno i da ne utječu štetno na značajke koje su radile ispravno prije dodavanja novih. Ukoliko se testiranje kompatibilnosti browsera ostavi za kraj projekta, popravljanje grešaka će biti skuplje i trajati će duže nego u slučaju konstantnog testiranja.

Tijek rada za ovu vrstu testiranja članak https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Introduction (13.01.2020.) dijeli se u četiri faze:

- Inicijalno planiranje;
- Razvoj;
- Testiranje;
- Popravci.

Inicijalno planiranje

U ovoj fazi testiranja kompatibilnosti browsera, obavlja se nekoliko sastanaka s klijentom kako bi se odredilo kako rješenje mora izgledati, koje funkcionalnosti će imati i slično. Također, određuje se koliko će trajati izrada rješenja, te koliko će cjelokupni projekt biti plaćen. Problemi u nekompatibilnosti rješenja u različitim browserima mogu jako utjecati na ovaj način planiranja.

Jednom kad je dogovoren popis funkcionalnosti i tehnologija koje će se koristiti pri izradi

rješenja, potrebno je procijeniti koja je ciljana publika za to rješenje. U kontekstu kompatibilnosti browsera, naravno, potrebno je odrediti kojim se browserima ciljana publika koristi. Moguće je da podatke o korištenju browsera od strane ciljane publike posjeduje klijent zbog prethodnih istraživanja obavljenih na drugim rješenjima koja posjeduje. U suprotnom je potrebno podatke potražiti u drugim izvorima kao što su statistika konkurenata ili država u kojima će se rješenje koristiti. Nakon analiziranja podataka, potrebno je ponovno proći kroz popis funkcionalnosti i odrediti da li ih je moguće sve implementirati u browserima koje ciljana publika koristi, a ako u određenim browserima to nije slučaj, potrebno je osigurati funkcionalno rješenje bez funkcionalnosti koje taj browser ne podržava.

Razvoj

Kroz fazu razvoja, potrebno je rješenje podijeliti u module (naslovna stranica, kontakt stranica, stranica s proizvodima…). Svrha te podjele je primjena neke od sljedećih strategija za razvoj rješenja za različite browsere:

- Potrebno je osigurati da sve funkcionalnosti funkcioniraju što je sličnije moguće u svim ciljanim browserima. Kako bi se ovo ostvarilo, moguće je da će biti potrebno pisanje različitog koda za različite browsere.
- Prihvaćanje da neke funkcionalnosti neće raditi jednako na svim browserima i osiguravanje sličnih, prihvatljivih rješenja u browserima koji ne podržavaju te funkcionalnosti.
- Prihvaćanje da rješenje neće funkcionirati na nekim starijim browserima. Ovo se radi uz suglasnost klijenta.

Uobičajeno je da se za vrijeme razvoja primjenjuje kombinacija svih navedenih strategija.

Testiranje

Po završetku svake faze implementacije, potrebno je testirati nove funkcionalnosti kako bi se osigurali da ne postoje problemi u kodu koji stopiraju rad rješenja i popraviti iste ako postoje. U ovu svrhu potrebno je:

- Testirati rješenje u nekoliko stabilnih browsera (Firefox, Safari, Chrome…);
- Obaviti jednostavne testove pristupačnosti (korištenje rješenja samo putem tipkovnice ili čitača ekrana);

- Testirati na mobilnoj platformi (Android/iOS).

Nakon toga, potrebno je proširiti listu browsera na cjelovitu listu ciljane publike i popisivati probleme među browserima:

- Testiranje zadnjih promjena na svim dostupnim browserima;
- Testiranje na uobičajenim browserima za mobilne uređaje i tablete (iOS Safari, Chrome, Firefox…).

Ako je moguće, testiranja je potrebno obaviti na fizičkim uređajima. Ukoliko to nije opcija, moguće je koristiti emulatore i virtualne mašine. Korištenje emulatora je praktično iz razloga što, na primjer, Windows ne dopušta simultano korištenje više verzija na istom računalu, a virtualna mašina to simulira. Testiranje unutar grupe korisnika je također opcija. U ovom slučaju, testeri postaju osobe izvan razvojnog tima (obitelj, prijatelji, kolege ili plaćeni profesionalni testeri). Ipak, najrazumnije rješenje je korištenje automatizacijskih alata, pogotovo u slučaju velikih projekata.

Popravci

Jednom kad je greška otkrivena, potrebno ju je pokušati ispraviti. Pri tom je prvo potrebno što više suziti gdje se greška pojavljuje te saznati što više informacija od osobe koja prijavljuje grešku (koja platforma, uređaj, verzija browsera…). Zatim je potrebno isprobati rješenje pod različitim konfiguracijama i vidjeti kako se greška ponaša. Moguće je da greška nije nastala krivnjom razvojnog tima, već browsera. Takve greške se obično poprave kroz ažuriranja browsera. Međutim, ukoliko je greška posljedica propusta u razvoju rješenja, potrebno ju je popraviti. Detektiranje greške obavlja se na isti način kao kod detektiranja bilo koje druge greške na webu. Jednom kad je greška otkrivena, nije dobro direktno u kodu ispraviti problem jer bi to moglo uzrokovati prestankom rada koda u drugim browserima, već je potrebno pronaći alternativno rješenje za browser u kojem se greška pojavljuje. Nakon popravka greške, dobro je ponoviti proces testiranja kako bi se osiguralo da popravak nije nanio štetu na drugim mjestima u drugim browserima.

5.1.1 Alati za testiranje kompatibilnosti browsera

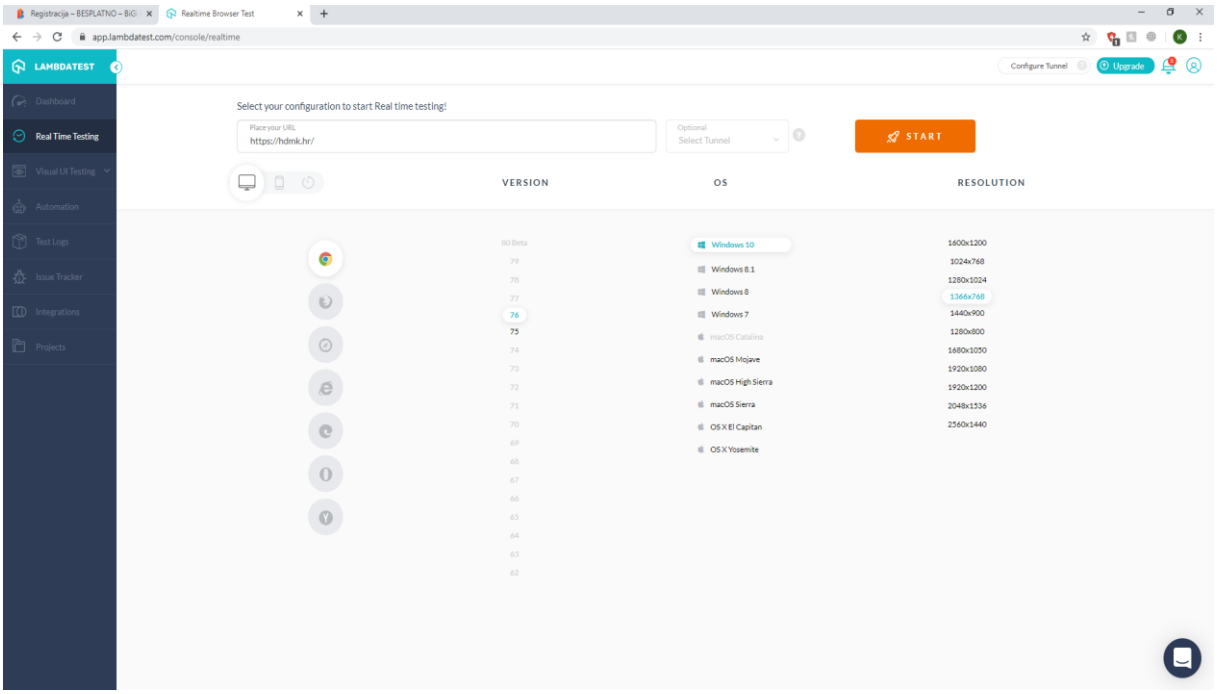
Testiranje kompatibilnosti browsera može zahtijevati mnogo vremena kako bi se ispravno obavilo i osiguralo kvalitetno korisničko iskustvo, pogotovo u slučaju velikih projekata. Kako

bi se proces skratio i uštedio novac potreban za obavljanje testiranja, dostupni su mnogi alati za obavljanje testiranja kompatibilnosti browsera. Neki od njih su:

1. **Lambda Test** – besplatan alat za testiranje kompatibilnosti browsera na cloudu. Jedan od boljih alata koji osiguravaju funkcioniranje web rješenja na gotovo svim desktop i mobilnim browserima današnjice. Ovaj alat testira rješenja na stvarnim browserima, pokrenutim na stvarnim uređajima. Korisnik može izabrati između preko 2000 desktop i mobilnih browsera i nudi mogućnost suradnje s timom kroz ugrađeni alat za praćenje. Nudi „Screenshot“ funkcionalnost koja pruža automatski generirane snimke zaslona cijele stranice web rješenja.
2. **Experitest** – Osnovne funkcije nudi besplatno, manualno testiranje po cijeni 49$ mjesečno, te automatizirano testiranje za 200$ mjesečno. Nudi preko 1000 desktop i mobilnih browsera putem clouda. Ima mogućnost interakcije s web rješenjem u stvarnom vremenu, te debugiranje koristeći platformu. Detektiranje i popravljanje problema responzivnog dizajna prije produkcije je također moguće. Kroz izvršavanje stotina paralelnih testova, kroz ovaj alat je moguće uštedjeti mnogo vremena.
3. **BrowserStack** – Po cijeni od 29$ mjesečno nudi Live verziju koja nudi neograničene snimke zaslona i responzivno testiranje, 129$ mjesečno za Automate Pro verziju koja nadodaje još i automatizaciju desktopa te snimke zaslona uz pomoć API-ja, te 199$ mjesečno za Automate Mobile verziju koja omogućuje i automatizaciju mobilnih uređaja. Korisnik putem ovog alata dobiva pristup za preko 1200 uređaja i browsera. Ima mogućnost testiranja web rješenja na različitim iOS i Android uređajima kroz različite browsere, a rezultati su ispravni i autentični. Također dozvoljava testiranje internih rješenja bez mijenjanja postavki i konfiguriranja.

5.1.2 Testiranje kompatibilnosti browsera u primjeni

Testiranje kompatibilnosti browsera obavljeno je pomoću alata Lambda Test. Taj alat je odabran iz razloga što je jedan od priznatih besplatnih alata te nudi preko 2000 desktop i mobilnih browsera. Početno sučelje alata Lambda Test prikazano je Slikom 12.



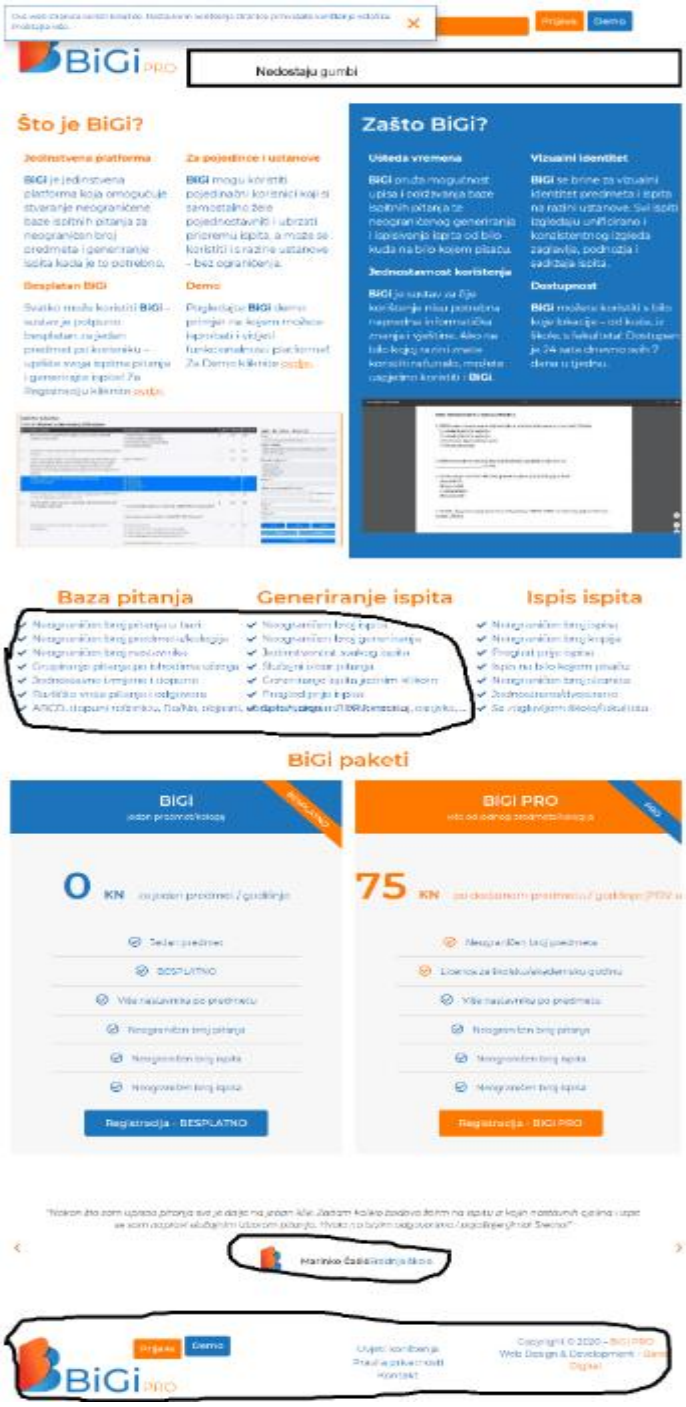
Slika 12: Početno sučelje alata Lambda Test

U svrhu ovog rada, obavljeno je testiranje pomoću snimki zaslona za razne browsere i razne uređaje. Tako je otkrivena pogreška u prikazu elemenata na uređaju iPhone 5 koji pokreće operacijski sustav iOS 6.0 (Slika 13).



Slika 13: Greška na iPhoneu 5

Slična greška, ali veće razine, se pojavljuje i na Internet Exploreru 10 za operacijski sustav Windows 8 gdje je neispravan poredak elemenata na stranici iznimno očit i narušava korisničko iskustvo rješenja (Slika 14).



Slika 14: Greška na Internet Exploreru 10

Iz priloženih grešaka, ali i nedostatka istih na novijim uređajima i browserima, da se zaključiti da je rješenje vrlo dobro prilagođeno novijim tehnologijama, dok na starim nailazi na probleme.

6. Testiranje performansi

„U kontekstu web developmenta, testiranje performansi obuhvaća korištenje alata za simuliranje načina rada rješenja pod određenim okolnostima. Kvantitativno testiranje performansi se bavi podatcima kao što je vrijeme odziva rješenje, dok se kvalitativno testiranje fokusira na skalabilnost, stabilnost i interoperabilnost. Dakle, kada kažemo da se testiraju performanse, iako mnogi odmah pomisle na brzinu aplikacije, to nije jedino mjerilo performansi. Iako je brzina učitavanja i brzina odziva definitivno nužna u današnje doba, nije dovoljno samo provjeriti rade li sve poveznice dovoljno brzo unutar rješenja. Moguće je da za vrijeme testiranja u fazi produkcije sve funkcionira savršeno, ali to ne znači da će se rješenje jednako ponašati ukoliko postane zatrpano korisnicima.

Kroz testiranje performansi rješenja, otkrivaju se problemi i poboljšavaju sveukupne performanse, a samim time i povećava korisničko iskustvo. Jedan od problema koje testiranje performansi može razotkriti je usko grlo *(engl. bottleneck).* Uska grla se mogu pojaviti ukoliko se dogodi nagli porast prometa kojim serveri nisu u stanju rukovati. Ukoliko se ne testira, ovakve stvari se obično saznaju kad je rješenje već pušteno korisnicima na korištenje.

Loša skalabilnost također može uvelike utjecati na performanse rješenja kroz usporavanja, greške i curenja memorije *(engl. memory leaks).*

Mogući su i problemi zbog ograničenja CPU-a i propusnosti *(engl. bandwith),* zbog čega se javlja potreba za realokacijom resursa ili ulaganjem u robusniju infrastrukturu.

Dobro je prikupiti što više informacija o korisnicima, kao što je način pristupa rješenju, kako bi se identificirali načini za poboljšanje korisničkog iskustva. (https://www.keycdn.com/blog/performance-testing 19.01.2020.)“

Kod testiranja performansi, prema prethodno referenciranom članku, mjeri se:

- **Vrijeme odziva** – količina vremena od izvođenja zahtjeva do odgovarajućeg odgovora. Vrijeme odziva može drastično varirati pod različitim uvjetima i akcijama.
- **Prosječno vrijeme učitavanja –** prosječno vrijeme odaziva za sve zahtjeve.
- **Maksimalno vrijeme odziva** – ukoliko je maksimalno vrijeme odziva puno duže od prosječnog vremena učitavanja, postoji problem.
- **Vrijeme čekanja** – referira se na vrijeme koje zahtjev provede u redu prije nego što ga se procesuira. Bitno je razlikovati vrijeme čekanja i vrijeme odziva jer ovise o različitim faktorima.
- **Broj zahtjeva po sekundi –** broj zahtjeva kojima se rukuje unutar jedne sekunde.

- **Upotreba memorije** – količina memorije potrebne za procesuiranje zahtjeva.
- **Stopa pogrešaka** – omjer pogrešaka u odnosu na zahtjeve.
- **Uspješne/neuspješne transakcije** – slično stopi pogrešaka, ali uzima u obzir druge faktore koji uzrokuju neuspjeh.
- **Broj istovremenih korisnika** – također zvana veličina opterećenja *(engl. load size).*
- **Propusnost** – dobar indikator kapaciteta rješenja. Postavljanje cilja propusnosti je dobar prvi korak prema popravljanju kapaciteta rješenja.

6.1 Koraci testiranja performansi

Postoji 6 koraka koje gore referencirani članak preporučuje pratiti kod testiranja performansi:

- **Postavljanje ciljeva** – potrebno je odlučiti koja mjerila su najbitnija korisnicima i testirati prema tome.
- **Planiranje scenarija** – potrebno je popisati specifične situacije u kojima se rješenje može naći. Na primjer, što će se dogoditi ako određeni broj korisnika pokuša obaviti istu akciju u isto vrijeme?
- **Priprema testnog okruženja** – potrebno je pobrinuti se da hardver i postavke mreže u kojima se testira dovoljno dobro simuliraju stvarne uvjete.
- **Testiranje i skupljanje podataka** – nakon provođenja testiranja pomoću alata, procjenjuju se rezultati.
- **Ponavljanje** – ponovno se provode testovi pod istim uvjetima kako bi se osigurala konzistentnost performansi. Nakon toga se testira pod drugačijim parametrima.

6.2 Testiranje opterećenja

„Testiranje opterećenja *(engl. load testing)* je vrsta nefunkcionalnog testiranja. Ovom vrstom testiranja se softver, simulirajući pristupanje korisnika rješenju u stvarnom životu, stavlja pod određeno opterećenje kako bi se shvatilo kako funkcionira pod normalnim, kao i pod ekstremnim opterećenjem. (http://tryqa.com/what-is-load-testing-in-software/ 20.01.2020.)“ Dakle, koristi se kako bi se osigurao zadovoljavajući rad rješenja u trenutcima kad mu mnogo korisnika pokuša pristupiti istodobno. Testiranjem opterećenja dobivamo odgovore na pitanja kao što su: „Je li rješenje održivo pod vršnim opterećenjem korisnika?“, „Koliko korisnika može istodobno pristupiti rješenju?“, „Koliko maksimalno posla rješenje može obavljati dok

mu performanse ne počnu padati?“ i slično. Testiranje opterećenja se uglavnom provodi na kraju SDLC-a kako bi se spriječili problemi s performansama što može klijenta dovesti do gubitka prihoda.

Neke od prednosti testiranja opterećenja su identificiranje problema vezanih za performanse i uskih grla prije puštanja rješenja u upotrebu, poboljšanje skalabilnosti rješenja u smislu baze podataka, softvera i mreže, minimalizacija rizika vezanog za stanke u radu rješenja, smanjenje cijene prestanka rada i rast zadovoljstva klijenta.

Nedostatci testiranja opterećenja mogu biti to što testeri moraju znati koristiti alate i u nekim slučajevima i programske jezike kako bi ga izvršili te cijena alata za testiranje.

Opterećenje možemo testirati tako da se:

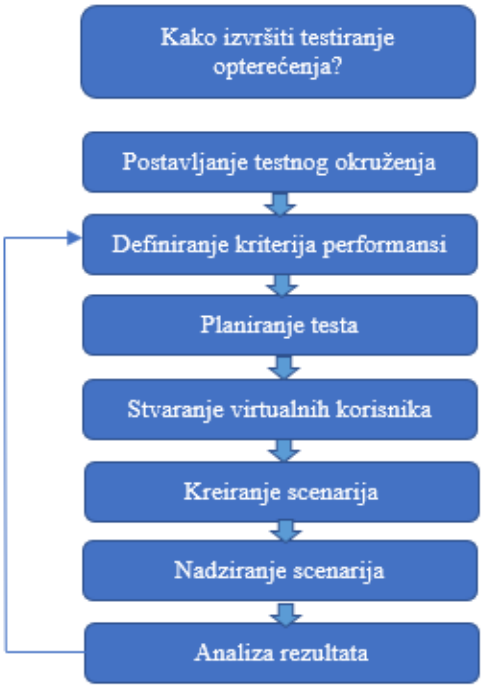
- Preuzima niz velikih datoteka s interneta;
- Pokreće više aplikacija na računalu ili serveru istodobno;
- Dodijeli mnogo poslova printeru u redu;
- Izloži server velikoj količini prometa;
- Neprekidno pišu i čitaju podatci s hard diska.

6.2.1 Proces testiranja opterećenja

Članak http://tryqa.com/what-is-load-testing-in-software/ (20.01.2020.) navodi kako proces testiranja opterećenja sadrži (Slika 15):

1. **Postavljanje testnog okruženja** – testno okruženje bi trebalo biti postavljeno što sličnije produkcijskom po pitanju hardvera, mreže, softvera i specifikacija.
2. **Definiranje kriterija performansi** – definiraju se mjerila uspješnosti performansi rješenja. Ovo može uključivati definiranje prihvatljivih granica vremena odgovora, transakcija, itd.
3. **Planiranje testa** – uključuje jasno definiranje plana testa opterećenja i postavljanje prikladnog okruženja u kojem će se test obaviti.
4. **Stvaranje virtualnih korisnika** – uključuje stvaranje skripti virtualnih korisnika koje sadrže razne zadatke koje virtualni korisnici trebaju izvesti.
5. **Kreiranje scenarija** – scenarij je kombinacija strojeva, skripti i virtualnih korisnika koji se pokreću za vrijeme testiranja. Postoje dvije vrste scenarija: manualni i orijentirani cilju.
6. **Pokretanje scenarija** – opterećenje servera se simulira pokretanjem više virtualnih korisnika koji simultano obavljaju zadatke. Prije pokretanja scenarija, potrebno ga je konfigurirati i namjestiti raspored obavljanja zadataka.

7. **Nadziranje scenarija** – scenariji se uglavnom nadziru koristeći online monitore alata za testiranje koji prate performanse rješenja.
8. **Analiza rezultata** - najvažniji korak u procesu testiranja opterećenja. Tester analizira performanse koristeći grafove i izvješća generirane tijekom izvršavanja scenarija. Test opterećenja je dobro ponoviti nakon popravljanja pronađenih problema.



Slika 15: Proces testiranja opterećenja

6.3 Testiranje otpornosti na stres

„Testiranje otpornosti na stres *(engl. stress testing)* je nefunkcionalni način testiranja kroz koji se određuje jesu li performanse rješenja zadovoljavajuće kad se na njega postavi ekstremna količina opterećenja. Ovom vrstom testiranja se osigurava da, u situacijama kad se rješenje nađe u abnormalnim uvjetima, greške u radu, kad se i pojave, postanu zanemarive. Iako je testiranje otpornosti na stres veoma važno, u većini projekata se ne obavlja do samog kraja SDLC-a, što može ostaviti ozbiljne posljedice ukoliko postoje problemi s performansama unutar rješenja. (http://tryqa.com/what-is-stress-testing-in-software/ 21.02.2020.)“

Testiranje otpornosti na stres uključuje forsiranje rješenja do točke raspada kako bi se promatralo rezultate. Ono je način utvrđivanja stabilnosti rješenja. Stavlja veći naglasak na robusnost, dostupnost i rukovanje greškama pod velikim opterećenjem, nego na ponašanje rješenja u normalnim uvjetima. Ciljevi ovakvog testiranja su osigurati da se softver ne ruši ni u

slučajevima kad se pokreće na nedovoljnim računalnim resursima (memorija, mreža, prostor na disku), a čak i kada se sruši, da se dovoljno brzo oporavi.
Kod testiranja otpornosti na stres, prema navedenom izvoru, mjeri se:

- **Prosječno vrijeme odgovora transakcije –** prosječno vrijeme potrebno za izvođenje transakcije.
- **Ukupan broj transakcija po sekundi** – ukupan broj transakcija koje su prošle, nisu prošle i koje su stale za vrijeme izvođenja testnog scenarija.
- **Transakcije po sekundi** – broj prolaza, padova i prestanaka rada za svaku transakciju u sekundi..
- **Vrijeme odgovora transakcije (pod opterećenjem)** – vrijeme odgovora vezano za broj virtualnih korisnika u nekom trenutku izvođenja scenarija.
- **Prosječan broj grešaka –** prosječan broj grešaka koji se dogode po sekundi izvođenja scenarija.
- **Broj HTTP zahtjeva od strane virtualnih korisnika za vrijeme izvođenja scenarija.**

6.3.1 Tipovi testiranja otpornosti na stres

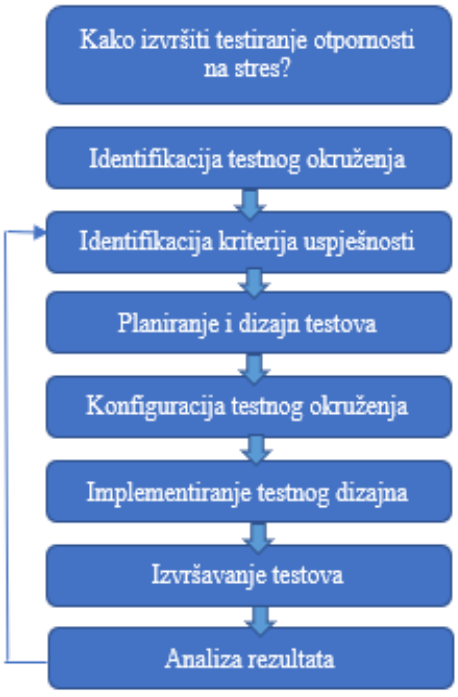
Članak http://tryqa.com/what-is-stress-testing-in-software/ (21.02.2020.) navodi sljedeće tipove testiranja otpornosti na stres:

1. **Distribuirano** – kod distribuiranog testiranja otpornosti na stres, testiraju se svi klijenti povezani na server.
2. **Transakcijsko** – koristi se kako bi se testirala otpornost na stres transakcija koje se događaju među aplikacijama. Svrha transakcijskog testiranja je optimiziranje sustava.
3. **Aplikacijsko** – obično se koristi kako bi se otkrili defekti vezani za uska grla u performansama, problemima s mrežom i blokadi podataka.
4. **Istraživačko** – u ovom slučaju, rješenje se izlaže abnormalnim uvjetima za koje nije vjerojatno da će se dogoditi u stvarnom životu. Na primjer: ekstremno velik broj korisnika se pokušava prijaviti, baza podataka se ugasi kad je rješenje pokuša dohvatiti iz front enda, dodavanje ekstremne količine podataka u bazu.
5. **Sistematsko** – koristi se kako bi se testirao velik broj sustava koji se pokreću na serveru. Omogućuje timu testera da otkriju neispravnosti gdje podatci jednog softvera blokiraju podatke drugog.

6.3.2 Proces testiranja otpornosti na stres

Prema prethodno navedenom izvoru, proces testiranja otpornosti na stres sadrži (Slika 16):

1. **Identifikaciju testnog okruženja** – određivanje hardvera, softvera, mreže, konfiguracije i alata kojima će se izvoditi testiranje.
2. **Identifikaciju kriterija uspješnosti performansi** – klasificiranje mjerila koja se analizira po završetku testiranja i određivanje kriterija njihove uspješnosti.
3. **Planiranje i dizajn testova otpornosti na stres** – stvaranje plana, identificiranje testnih scenarija…
4. **Konfiguraciju testnog okruženja** – priprema testnog okruženja, alata i resursa potrebnih za izvođenje svih scenarija.
5. **Implementiranje testnog dizajna** – razvoj testova otpornosti na stres u skladu s najboljim praksama testnog dizajna.
6. **Izvršavanje testova** – pokretanje i nadziranje testova te validiranje testova, dobivenih podataka i rezultata.
7. **Analizu rezultata** – kad su sva mjerila unutar prihvatljivih okvira, sve je u redu i sve željene informacije su prikupljene.



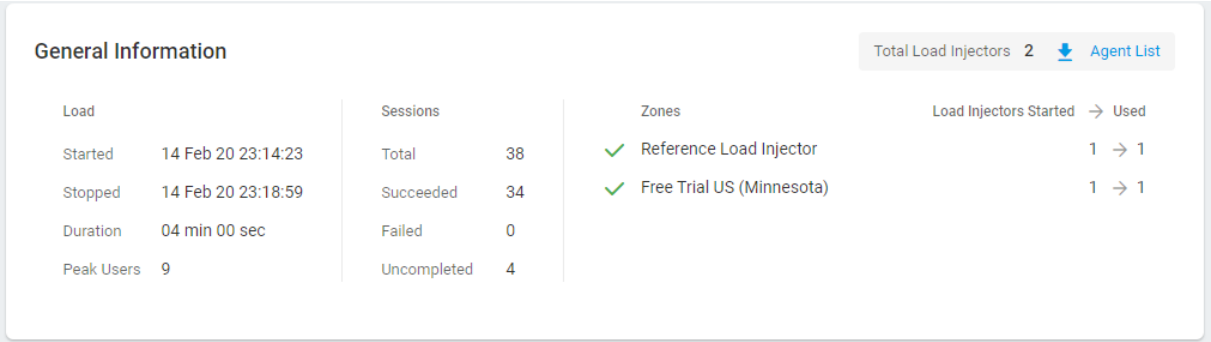
Slika 16: Proces testiranja otpornosti na stres

6.4 Alati za testiranje performansi

1. **WebLOAD** – Najčešće se koristi u velikim tvrtkama koje imaju potrebu za kompleksnim testovima te njihova rješenja posjećuje velik broj korisnika. Omogućuje testiranje opterećenja, kao i testiranje otpornosti na stres na bilo kojem web rješenju generirajući korisničko opterećenje na cloudu i lokalnim računalima. Prednosti WebLOADa su njegova fleksibilnost i jednostavnost korištenja. Pruža brzo definiranje potrebnih testova s mogućnostima poput snimanja baziranog na DOM-u, automatska korelacija i JavaScript. Također omogućuje jasne analize performansi aplikacije ističući probleme i uska grla. Integriran je s raznim DevOps alatima poput Jenkinsa i Seleniuma. Postoji besplatno izdanje i profesionalno izdanje koje se plaća.
2. **LoadView** – Alat koji omogućuje testiranje opterećenja i testiranje otpornosti na stres. LoadView, za razliku od većine drugih alata, provodi testiranja u stvarnim browserima, što omogućuje iznimno točne rezultate. Plaća se ono što se koristi i nisu potrebni ugovori. 100% je baziran na cloudu i skalabilan. Neke od njegovih naprednih karakteristika su skriptiranje pokaži i klikni *(engl. point and click scripting),* globalna infrastruktura bazirana na cloudu i testiranje u stvarnim browserima.
3. **LoadNinja** - omogućuje brzo stvaranje sofisticiranih testova opterećenja bez skripti, smanjuje vrijeme testiranja za 50% i zamjenjuje emulatore stvarnim browserima. Kroz ovaj alat se može lako zabilježiti interakcije od strane klijenta, debugirati u stvarnom vremenu i identificirati probleme s performansama u veoma kratkom vremenu. Karakteristike LoadNinje omogućuju testerima da povećaju količinu pokrivenosti rješenja testovima bez žrtvovanja kvalitete kroz uklanjanje truda koji iziskuje stvaranje skripti. Postoji Starter, Pro i Premium verzija.

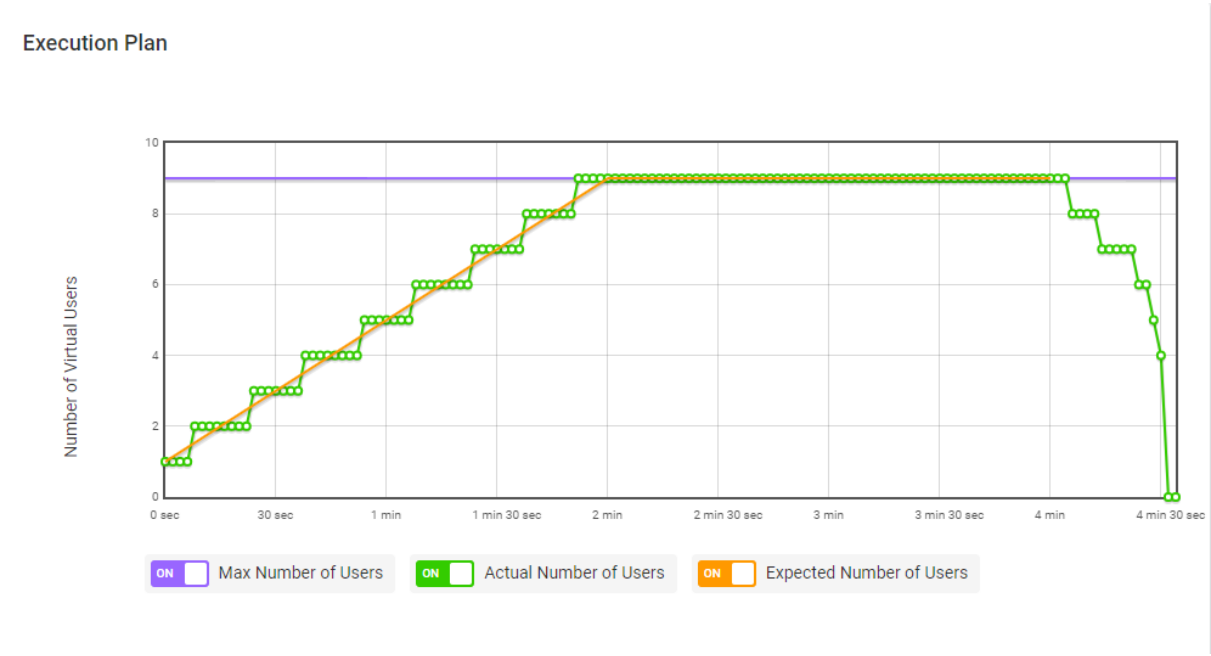
6.5 Testiranje performansi u primjeni

Testiranje performansi je obavljeno unutar alata LoadView koji omogućuje besplatno korištenje alata na 30 dana. Slika 17 prikazuje generalne informacije provedenog testiranja.



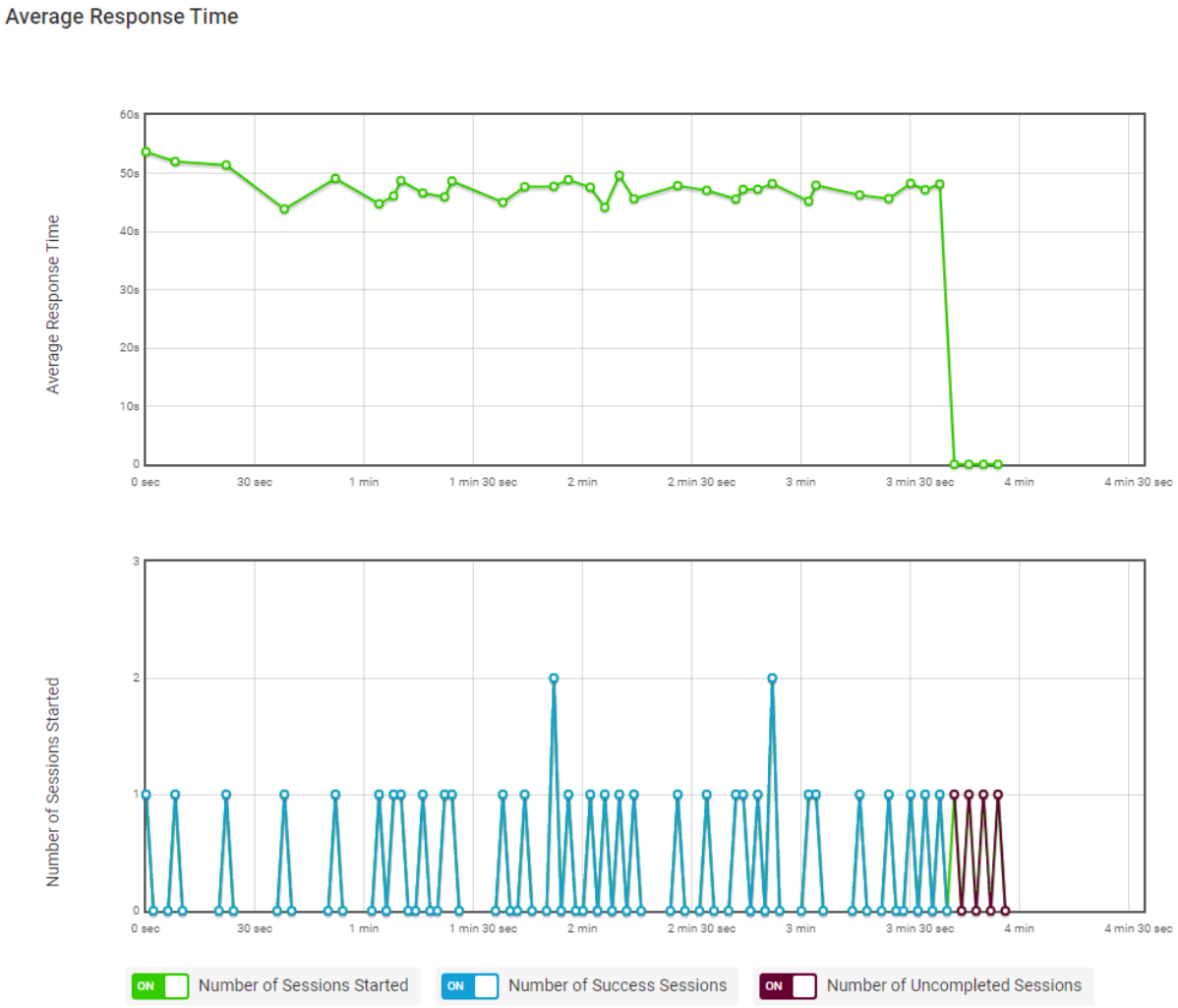
Slika 17: Generalne informacije testiranja performansi

Slika 18 prikazuje graf na kojem se vidi plan izvođenja testa, te porast broja korisnika unutar 4 minute.



Slika 18: Plan izvođenja

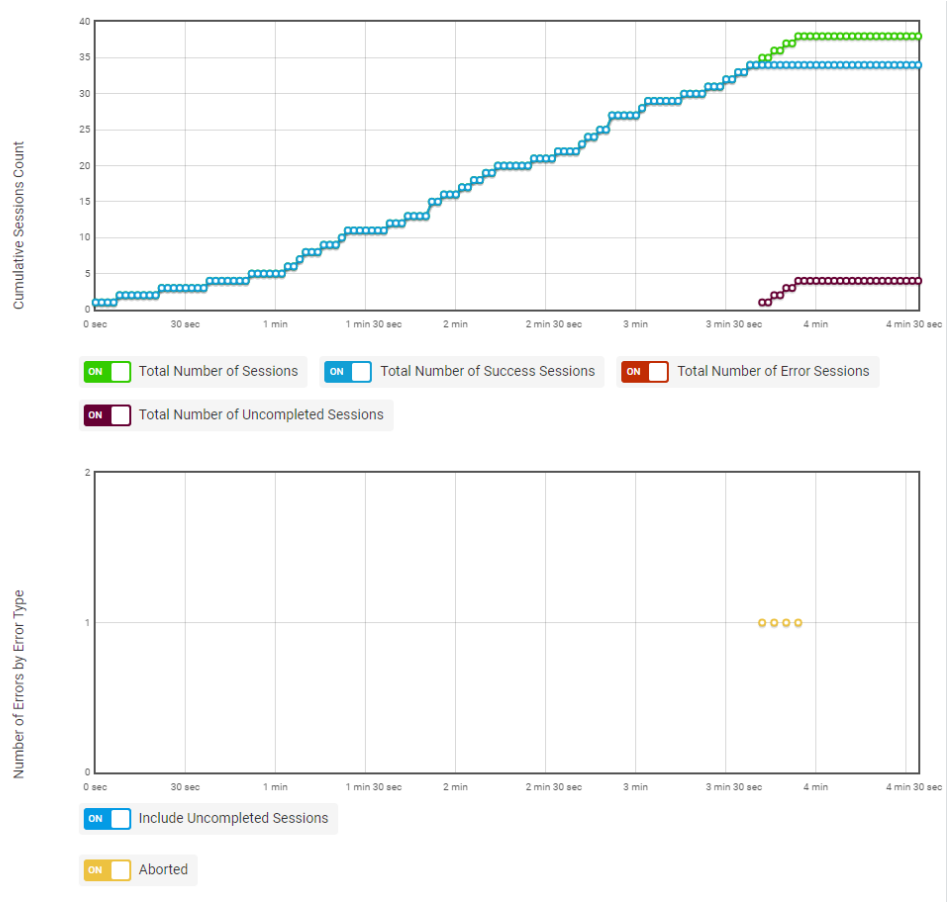
Slika 19 prikazuje grafove koji sadrže informacije o prosječnom vremenu odgovora i boju započetih sesija.



Slika 19: Vrijeme odgovora i broj sesija

Slika 20 prikazuje kumulativan broj sesija te broj grešaka za vrijeme izvođenja testa.

Prema informacijama dobivenim iz ovih grafova, da se zaključiti da rješenje na ovom uzorku daje zadovoljavajuće rezultate. Nažalost, zbog skupoće plaćene verzije alata i ograničenja besplatnih verzija, nije bilo moguće obaviti testiranje na većem uzorku korisnika.



Slika 20: Kumulativni broj sesija i broj grešaka

7. Sigurnosno testiranje

Sigurnost se u IT kontekstu definira kao očuvanje povjerljivosti, integriteta i dostupnosti informacija. Pri tom je povjerljivost svojstvo dostupnosti informacija samo autoriziranim osobama, subjektima ili procesima, integritet je svojstvo točnosti i cjelovitosti informacija, a dostupnost svojstvo pristupačnosti i upotrebljivosti informacija na zahtjev autoriziranog subjekta. „Sigurnosno testiranje je vrsta testiranja softvera kojom se otkrivaju ranjivosti, prijetnje i rizici unutar rješenja i na taj način se sprječavaju zlonamjerni napadi. Svrha sigurnosnog testiranja je identifikacija svih mogućih slabosti rješenja koje mogu uzrokovati gubitkom informacija, prihoda te ugleda osoba koje koriste rješenje ili rade na njemu. Cilj je pronaći sve prijetnje rješenju te izmjeriti potencijalnu ranjivost kako rješenje ne bi postalo izloženo ili prestalo raditi. (https://www.guru99.com/what-is-security-testing.html 23.01.2020.)“

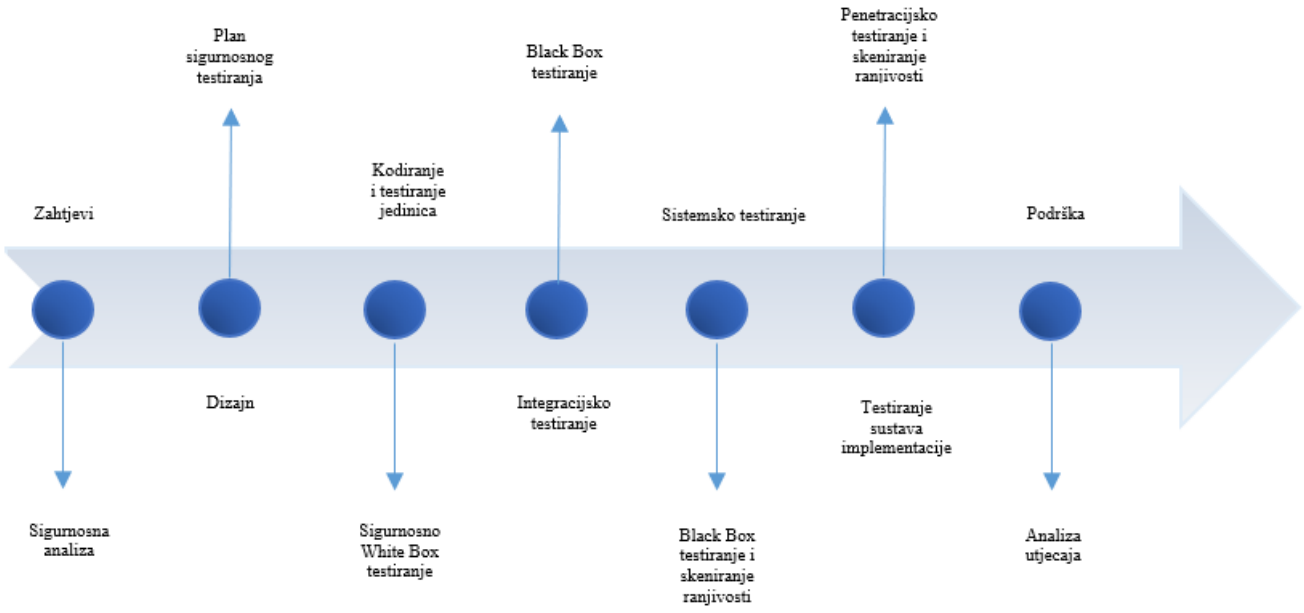
„Kroz zadnje desetljeće, web rješenja su zadobila povjerenje mnogih korisnika na račun sigurnosti. Bez sigurnosti se ne bi isplatilo imati rješenja kao što su webshopovi, bankarske aplikacije, aplikacije za trgovinu dionicama, razna rješenja za online plaćanje i slično. Ukoliko rješenje ne može zaštititi podatke koji se njime prenose, vjerojatnost da će ga netko koristiti pada na minimum. Stoga web developeri rješenja moraju učiniti otpornim na SQL injekcije, napade sirovom snagom *(engl. brute force attacks)* i XSS. (https://www.softwaretestinghelp.com/how-to-test-application-security-web-and-desktop-application-security-testing-techniques/ 23.01.2020.)“

Postoji sedam glavnih tipova sigurnosnog testiranja (https://www.geeksforgeeks.org/software-testing-security-testing/ 23.01.2020.):

1. **Skeniranje ranjivosti** – radi se pomoću automatiziranih softvera kako bi se otkrili znakovi ranjivosti unutar rješenja.
2. **Sigurnosno skeniranje** – uključuje identifikaciju mrežnih i sistemskih slabosti te nudi rješenja za smanjenje rizika. Može se izvoditi ručno ili automatski.
3. **Penetracijsko testiranje** – simulira napad od strane zlonamjernog hakera. Uključuje analizu određenog sustava kako bi se provjerile potencijalne ranjivosti na pokušaje hakiranja.
4. **Procjena rizika** – analiziraju se sigurnosni rizici unutar organizacije. Rizici se klasificiraju u tri razine rizika: niska, srednja i visoka. Ovaj tip testiranja preporuča kontrole i mjere za smanjivanje rizika.

5. **Revizija sigurnosti** – unutarnja inspekcija rješenja i operacijskih sustava sa svrhom otkrivanja sigurnosnih propusta. Može se obavljati i kroz pregled koda liniju po liniju.
6. **Etičko hakiranje** – hakiranje sustava organizacije. Za razliku od malicioznih hakera kojima je cilj ostvariti dobit na ovaj način, kod etičkog hakiranja je namjera razotkriti mane sustava kako bi se mogle popraviti.
7. **Procjena stanja *(engl. posture assessment)*** – kombinira se sigurnosno skeniranje, etičko hakiranje i procjena rizika kako bi se procijenilo sveukupno sigurnosno stanje.

„Jasno je da je sigurnosni testovi koštaju više ukoliko se odgađaju za fazu implementacije rješenja. Stoga je bitno početi testirati za vrijeme SDLC-a, u ranijim fazama. (https://www.guru99.com/what-is-security-testing.html 23.01.2020.)“ Dolje priložene Slika 21 i Tablica 1 opisuju testiranje sigurnosti po fazama SDLC-a.



Slika 21: Testiranje sigurnosti po fazama SDLC-a

Tablica 1 Sigurnosni procesi po fazama SDLC-a

Faze SDLC-a	Sigurnosni procesi
Zahtjevi	Sigurnosna analiza zahtjeva i provjera za slučajevima zlouporabe.
Dizajn	Analiza sigurnosnih rizika za dizajn. Razvoj testnog plana, uključujući sigurnosne testove.
Kodiranje i testiranje jedinica	Statičko i dinamičko testiranje sigurnosti. Testiranje bijele kutije *(engl. white box testing)*
Integracijsko testiranje	Testiranje crne kutije *(engl. black box testing).*
Testiranje sustava	Testiranje crne kutije i skeniranje ranjivosti.
Implementacija	Penetracijsko testiranje i skeniranje ranjivosti.
Podrška	Analiza utjecaja popravaka

7.1 Tehnike testiranja sigurnosti

Prema članku https://www.softwaretestinghelp.com/how-to-test-application-security-web-and-desktop-application-security-testing-techniques/ (23.01.2020.), tehnike testiranja sigurnosti su sljedeće:

1. **Pristup rješenju** – sigurnost pristupa rješenju implementira se kroz upravljanje ulogama i pravima. Često se to obavlja implicitno kroz pokrivanje funkcionalnosti. Primjer ograničenja pristupa i prava određene uloge može biti sustav jedne bolnice. Recepcionist te bolnice ne smije vidjeti rezultate laboratorijskih testiranja jer je njegov posao registracija pacijenata i raspored njihovih sastanaka s doktorima.
   Kako bi se ovo testiralo, tester bi trebao napraviti nekoliko korisničkih računa s

različitim ulogama i zatim provjeravati da li svaka od tih uloga ima pristup vlastitim modulima, formama i izbornicima. Ukoliko se pronađe neka nepravilnost, potrebno je zapisati da postoji problem. Dakle, testira se tko je korisnik koji koristi rješenje i što on smije raditi.

2. **Zaštita podataka** – bazira se na tri aspekta sigurnosti. Prvi nalaže da bi korisnik, za vrijeme korištenja rješenja trebao moći vidjeti i koristiti samo one podatke koji su njemu i namijenjene za korištenje. Ovo se također osigurava kroz upravljanje ulogama i pravima.

   Drugi aspekt je povezan s načinom pohrane podataka u bazu podataka. Svi povjerljivi podatci moraju biti kriptirani kako bi se osigurali. Ovo se posebno odnosi na lozinke, brojeve kreditnih kartica i druge ključne informacije.

   Treći aspekt je zapravo nastavak na drugi aspekt. U slučajevima kada dolazi do razmjene osjetljivih podataka, bilo između modula istog rješenja ili među dva različita rješenja, potrebno ih je osigurati na primjeren način. Dakle, povjerljivi podatci moraju biti kriptirani.

   Kako bi se ovo testiralo, potrebno je provjeriti da li su svi osjetljivi podatci u bazi kriptirani te da li se pri prijenosu tih podataka ispravno obavlja enkripcija i dekripcija. Također je potrebno osigurati da se povjerljivi podatci ne pojavljuju u adresnoj traci unutar browsera u nekom razumljivom formatu.

3. **Napad sirovom snagom** – uglavnom se radi pomoću određenih softverskih alata. Radi se tako da se koristi ispravan ID korisnika, a potom alat pokušava pogoditi lozinku koja je povezana s tim ID-em. Jednostavan primjer sigurnosne mjere protiv ovakvog napada je zaključavanje računa na neko vrijeme nakon što se lozinka ne pogodi određen broj puta (uglavnom je to 3).

   Kako bi se testirala otpornost na ovakve napade, tester mora potvrditi da postoji neki mehanizam zaključavanja računa, te da on radi ispravno. Dakle, tester se pokušava neuspješno prijaviti na određeni račun nekoliko puta i provjerava da li to može raditi neograničeno.

4. **SQL injekcija i XSS** – Konceptualno se ove dvije vrste napada slične, tako da se o njima raspravlja zajedno. Pristup je takav da maliciozna skripta pokušava manipulirati web rješenjem. Postoji nekoliko načina za obranu od ovih napada. Za sva polja za unošenje unutar web rješenja, količina znakova bi trebala biti definirana dovoljno mala kako bi se ograničio unos skripti unutar njih. Na primjer, prezime bi trebalo ograničiti na 30 znakova umjesto 255. Međutim, postoje i slučajevi kad je potrebno unijeti veliku

količinu teksta. U takvim slučajevima je potrebno zabraniti upotrebu bilo kakvih HTML tagova.

5. **Pristupne točke usluga** – bazira se na potrebi poslovnih suradnika da mogu na određenim mjestima unutar rješenja pristupiti jedni drugima. U većim sustavima, potrebno je osigurati da su ove pristupne točke dovoljno otvorene kako bi mogle primiti velik broj korisnika, ali i dovoljno sigurne kako bi se mogle nositi sa neželjenim pristupom. Dakle, potrebno je, na neki način, odvojiti željene od neželjenih korisnika. To se može dobiti prepoznavanjem IP adresa.
6. **Upravljanje sesijama** – web sesija je niz HTTP zahtjeva i odgovora povezanih s istim korisnikom. Testovi upravljanja sesijom provjeravaju da li se ispravno rukuje sesijama unutar web rješenja. Potrebno je testirati da li sesija istekne nakon određenog vremena, da li se po zatvaranju sesije događa odjava s rješenja, koliko traju kolačići *(engl. cookies)* te može li isti korisnik imati više sesija.
7. **Rukovanje greškama** – testiranje rukovanja greškama uključuje provjeru kodova greške i provjeru stack tracea.
8. **Specifične rizične fukncionalnosti** – dvije česte rizične funkcionalnosti su plaćanja i upload datoteka. Kod uploada datoteka potrebno je testirati jesu li zabranjene neželjene i maliciozne datoteke, a za plaćanja se testira sigurnost kriptografskog pohranjivanja, pogađanje lozinki i slično.

7.3 Alati za testiranje sigurnosti

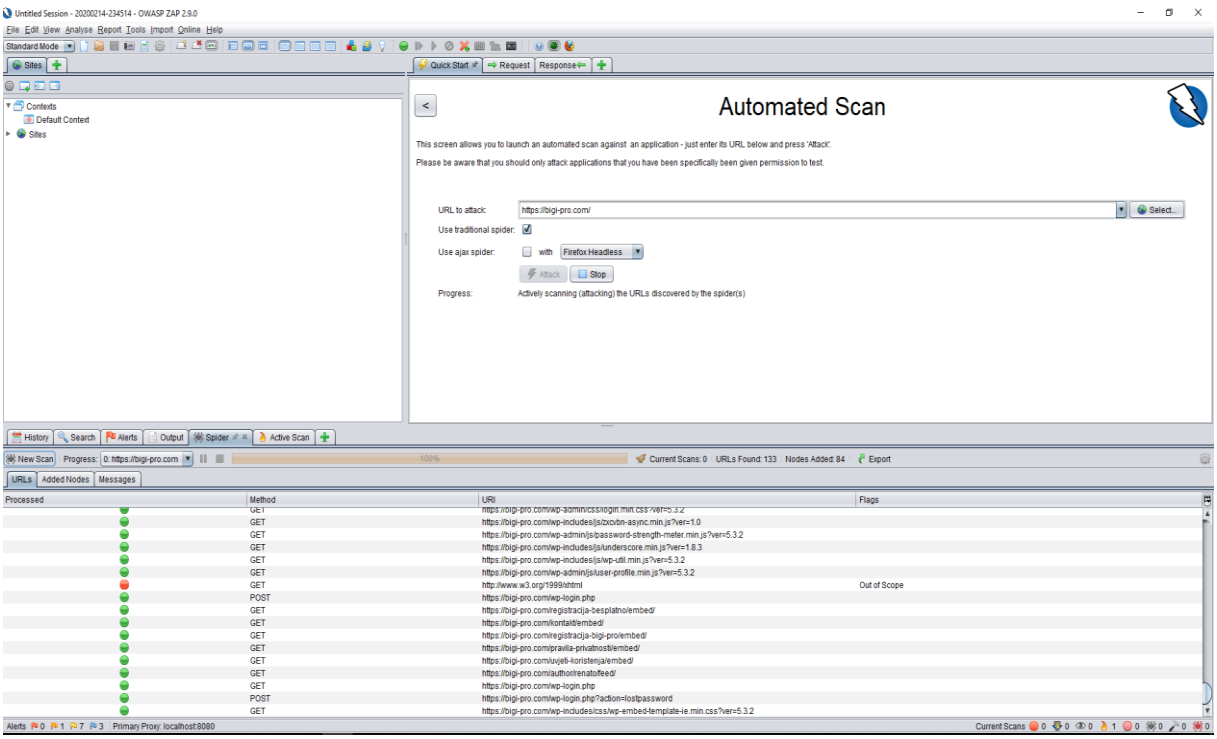
1. **Owasp** – The Open Web Application Security Project (OWASP) je međunarotna neprofitna organizacija fokusirana na poboljšanje sigurnosti softvera. Pruža više alata za penetracijsko testiranje softverskog okruženja i protokola:
    - Zed Attack Proxy (ZAP) – integrirani alat za penetracijsko testiranje.
    - Owasp Dependency Check – skenira zavisnosti rješenja i provjerava ranjivosti.
    - Owasp Web Testing Environment Project – kolekcija sigurnosnih alata i dokumentacija.
2. **WireShark** – alat za analizu mreže. Dohvaća pakete u stvarnom vremenu i prikazuje ih u čitljivom formatu. Wireshark je analizator mrežnih paketa i omogućuje detalje o mrežnim protokolima, dekripciji, informacijama o paketima i slično. Open source je i samo neki od operacijskih sustava na kojima se može koristiti su Linux, Windows, OS

X, Solaris, NetBSD i FreeBSD. Informacije koje pruža WireShark se mogu prikazati kroz GUI ili TTY mod.

3. **W3af** – web rješenje koje sadrži tri tipa pluginova: discovery, audit i attack koji kumuniciraju međusobno i traže ranjivosti unutar rješenja.

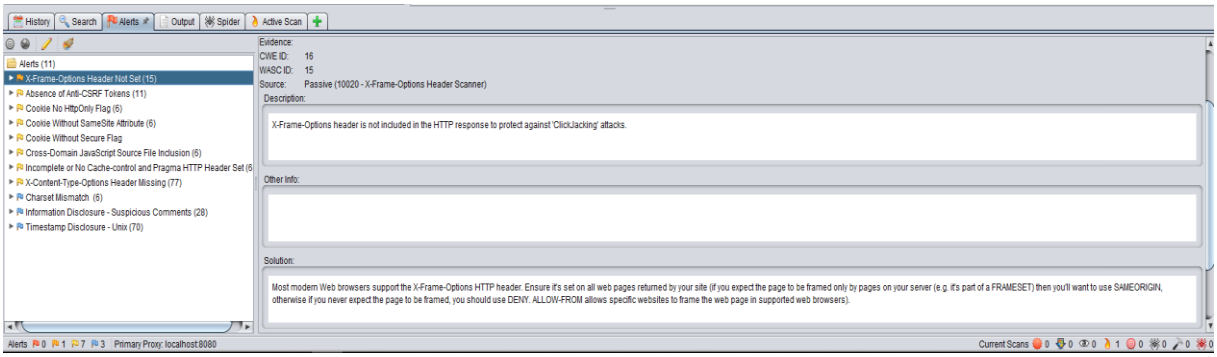
7.4 Sigurnosno testiranje u primjeni

Sigurnosno testiranje provedeno je kroz besplatan alat OWASP ZAP koji se nalazi na listi OWASP Top 22. Slika 20 prikazuje sučelje alata OWASP ZAP nakon izvršenog skeniranja web rješenja.



Slika 22: Skeniranje sigurnosti rješenja

Slika 23 prikazuje uzbune dobivene nakon skeniranja rješenja. Za svaku dobivenu uzbunu, alat nudi opis, dodatne informacije i rješenje koje je potrebno poduzeti kako bi se propusti sanirali. Iz analize uzbuna se tako da zaključiti da je rješenje osjetljivo na „*Click Jacking*“ napade (tehnika zavaravanja korisnika da klikne nešto zlonamjerno, iako korisnik misli da radi ispravno), XSS i „*MIME sniffing*“ (njuškanje sadržaja – pregledavanje sadržaja bajt toka kako bi se zaključio format datoteke podataka unutar njega).



Slika 23: Sigurnosne uzbune

8. Zaključak

U ovom radu obrađena je tema važnosti testiranja web rješenja. Na primjerima gubitaka aerodroma Heathrow, rušenja zrakoplova i neugodnosti oko neispravnosti zatvorskog sustava je prikazana važnost testiranja, ne samo za financijske, već i za društvene aspekte. Kako bi se takve nesreće spriječile, potrebno je znati kako ispravno testirati rješenja,

Cjelovito testiranje web rješenja se provodi kroz sljedeće vrste testiranja: funkcionalno testiranje, testiranje upotrebljivosti, testiranje sučelja, testiranje kompatibilnosti, testiranje performansi i testiranje sigurnosti. Kako bi se maksimalno umanjila vjerojatnost pojave grešaka u rješenju, potrebno je provesti sve navedene vrste primjenjujući principe uspješnog testiranja. Principi kojima se postiže uspješno testiranje nalažu da je nemoguće obaviti iscrpno testiranje, što znači da nije moguće pokriti apsolutno sve scenarije pod kojima se web rješenje može pokrenuti, a neke stvari se namjerno ne testiraju zbog opsega i cijene. Na testiranje se primjenjuje Paretov zakon koji nalaže da je 80% problema unutar 20% modula. To, naravno, ne znači da se 80% modula smije zanemariti, već nam ukazuje na što se treba dodatno fokusirati. Treba uzeti u obzir i paradoks pesticida prema kojem se, konstantnim ponavljanjem istih testova ne otkrivaju nove mane rješenja. Također je bitno znati da testiranje ukazuje na postojanje mana, ne na nedostatak istih, a nedostatak grešaka može dovesti do zablude ukoliko se testiraju pogrešni moduli. Testiranje je potrebno početi provoditi što ranije unutar SDLC-a kako bi se smanjio trošak vremena i novca potrebnog za kasnije popravke. Bitno je znati i da testiranje ovisi o kontekstu, pa tako se ne testira na jednak način video igra i web shop.

Provođenjem svih vrsta testiranja opisanih u ovom radu i primjenjivanjem principa uspješnog testiranja, ne stvara se garancija da se neka greška neće pojaviti u nekom trenutku izvođenja rješenja zbog toga što, koliko god se rješenje testira, nikad se ne mogu pokriti svi scenariji izvođenja. Međutim, vjerojatnost za pojavom greške se drastično povećava ukoliko se testiranje ne obavi ispravno, a još više ukoliko u potpunosti izostane. Iako se rezultati testiranja možda ne vide u svakodnevnom životu, ono definitivno služi kao mjera sa sprječavanje velikih gubitaka.

Popis kratica

SDLC	Software development Life Cycle	životni ciklus razvoja softvera
API	Application Programming Interface	sučelje za programiranje aplikacija
CPU	Central Processing Unit	središnja jedinica za obradbu
DOM	Document Object Model	model objekta dokumenta
XSS	Cross Site Scripting	međustranično skriptiranje
GUI	Graphic User Interface	grafičko korisničko sučelje
TTY	Teletypewriter	teleprinter

Popis slika

Slika 1: Greška zatvorskog sustava 5
Slika 2: Naslovna stranica rješenja BiGi PRO 12
Slika 3: Prikaz podnožja BiGi PRO s gumbima "Prijava" i "Demo" 13
Slika 4: Greška na tekstnom okviru - registracijska forma 13
Slika 5: Stranica za upravljanje adresama unutar browsera 14
Slika 6: Proces testiranja upotrebljivosti 16
Slika 7: Prikaz gumba zaglavlja BiGi PRO 17
Slika 8: Ispravno popunjena kontakt forma 19
Slika 9: Ispravno popunjena registracijska forma 20
Slika 10: Primljeni e-mail s podatcima kontakt forme 20
Slika 11: Primljeni e-mail s podatcima registracijske forme 21
Slika 12: Početno sučelje alata Lambda Test 28
Slika 13: Greška na iPhoneu 5 28
Slika 14: Greška na Internet Exploreru 10 29
Slika 15: Proces testiranja opterećenja 33
Slika 16: Proces testiranja otpornosti na stres 35
Slika 17: Generalne informacije testiranja performansi 37
Slika 18: Plan izvođenja 37
Slika 19: Vrijeme odgovora i broj sesija 38
Slika 20: Kumulativni broj sesija i broj grešaka 39
Slika 21: Testiranje sigurnosti po fazama SDLC-a 41
Slika 22: Skeniranje sigurnosti rješenja 45
Slika 23: Sigurnosne uzbune 46

Popis tablica

Tablica 1 Sigurnosni procesi po fazama SDLC-a 42

Literatura

[1] Paul Ammann; Jeff Offutt; Introduction to software testing (2008.);

[2] Alexander Tarlinder; Developer Testing: Building Quality Into Software (2016.)

[3] Jonathan Rasmusson: The Way of the Web Tester: A Beginner's Guide to Automating Tests (2016.)

[4] Mike Andrews; James A. Whittaker; How To Break Web Software: Functional and Security Testing of Web Applications and Web Services (2006.)

[5] Hung Q Nguyen; Testing Applicatons on the Web (2001.)

[6] https://www.javatpoint.com/software-testing-introduction (12.10.2019.)

[7] https://usersnap.com/blog/web-application-testing/ (12.10.2019.)

[8] https://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/ (12.10.2019)

[9] https://www.softwaretestinghelp.com/web-application-testing/ (12.10.2019.)

[10] http://www.poslovni.hr/strane-kompanije/novi-terminal-londonskog-heathrowa-aerodromu-donio-dodatne-probleme-75596 (21.11.2019.)

[11]https://www.injurytriallawyer.com/blog/where-did-the-department-of-corrections-go-wrong-.cfm (21.11.2019.)

[12]https://www.bloomberg.com/news/articles/2019-07-27/latest-737-max-fault-that-alarmed-test-pilots-rooted-in-software (21.11.2019.)

[13]https://www.guru99.com/software-testing-seven-principles.html (22.11.2019.)

[14] https://www.softwaretestinghelp.com/guide-to-functional-testing/ (24.11.2019.)

[15]http://softwaretestingfundamentals.com/functional-testing/ (24.11.2019.)

[16]http://softwaretestingfundamentals.com/unit-testing/ (24.11.2019.)

[17]http://softwaretestingfundamentals.com/smoke-testing/ (24.11.2019.)

[18] https://www.geeksforgeeks.org/sanity-testing-software-testing/ (2.12.2019.)

[19] https://www.guru99.com/regression-testing.html (2.12.2019.)

[20] http://softwaretestingfundamentals.com/integration-testing/ (2.12.2019.)

[21] http://softwaretestingfundamentals.com/system-testing/ (2.12.2019.)

[22] https://www.experienceux.co.uk/faqs/what-is-usability-testing/ (10.12.2019.)

[23] https://www.guru99.com/usability-testing-tutorial.html (10.12.2019.)

[24] https://www.educba.com/interface-testing/ (10.12.2019.)

[25] http://tryqa.com/what-is-compatibility-testing-in-software/ (12.12.2019.)

[26] https://www.invensis.net/blog/it/what-is-compatibility-testing/ (12.12.2019.)

[27] https://www.testbytes.net/blog/compatibility-testing/ (12.12.2019.)

[28] https://www.softwaretestinghelp.com/software-compatibility-testing/ (12.12.2019.)

[29] https://www.browserstack.com/guide/cross-browser-compatibility-testing-beyond-chrome (13.01.2019.)

[30]https://developer.mozilla.org/enUS/docs/Learn/Tools_and_testing/Cross_browser_testing/Introduction (19.01.2020.)

[31] https://www.keycdn.com/blog/performance-testing (19.01.2020.)

[32] http://tryqa.com/what-is-load-testing-in-software/ (20.01.2020.)

[33] http://tryqa.com/what-is-stress-testing-in-software/ (21.01.2020.)

[34] https://www.softwaretestinghelp.com/performance-testing-tools-load-testing-tools/ (22.01.2020.)

[35] https://www.softwaretestinghelp.com/how-to-test-application-security-web-and-desktop-application-security-testing-techniques/ (23.01.2020.)

[36] https://www.geeksforgeeks.org/software-testing-security-testing/ (23.01.2020.)

[37] https://www.guru99.com/what-is-security-testing.html (23.01.2020.)