Višeplatformsko testiranje web i mobilnih aplikacija

Lekić, Matija

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij

VIŠEPLATFORMSKO TESTIRANJE WEB I MOBILNIH APLIKACIJA

Diplomski rad

Matija Lekić

Osijek, 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 06.09.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Matija Lekić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 779 R, 09.10.2015.
OIB studenta:	23263813681
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	Bernardin Katić, dipl.ing.
Predsjednik Povjerenstva:	Doc.dr.sc. Alfonzo Baumgartner
Član Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Naslov diplomskog rada:	Višeplatformsko testiranje web i mobilnih aplikacija
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Osiguranje kvalitete web i mobilnih aplikacija kojima je zbog zahtjeva tržišta nužna česta nadogradnja veliki je izazov. Cilj ovog diplomskog rada je osigurati ispravan rad aplikacija na velikom broju različitih mobilnih uređaja, operacijskih sustava i web preglednika. U teorijskom dijelu rada potrebno je opisati projektnu metodologiju, načine osiguravanja kvalitete rješenja, razraditi strategiju testiranja, te automatiziranja testiranja u opisanim uvjetima. U praktičnom dijelu rada treba istražiti raspoložive platforme na kojima je moguće automatizirati testiranje aplikacija. Nadalje, treba razviti metodologiju definiranja raspona uređaja i sustava za koje je potrebno obaviti testiranje, te izvršiti i analizirati automatiziranje testiranja za prikladno stvarno rješenje. (sumentor: Bernard Katić, dipl.ing., tvrtka: Hammer d.o.o.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	06.09.2017.

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

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2017.

Ime i prezime studenta:	Matija Lekić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 779 R, 09.10.2015.
Ephorus podudaranje [%]:	1

Ovom izjavom izjavljujem da je rad pod nazivom**: Višeplatformsko testiranje web i mobilnih aplikacija**

izrađen pod vodstvom mentora Prof.dr.sc. Goran Martinović

i sumentora

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

Potpis studenta:

Sadržaj

1 UVOD

Sastavni dio procesa izrade programskog rješenja je testiranje istog. Štoviše, posao testiranja funkcionalnosti, dizajna te pouzdanosti programske podrške zauzima jednaku ulogu i važnost kao i stvaranje idejnog rješenja zajedno s implementacijom u obliku programskog koda. U moru tehnologija i načina izrade računalnih aplikacija posao testera ne podrazumijeva jedinstveni skup vještina i tehnika.

S rastućim trendom razvoja višestrukih tehnologija za izradu mrežnih aplikacija raste i mogućnost izrade vrlo složenih programskih rješenja. Zbog toga je izuzetno važno provoditi redovito i temeljito testiranje takvih rješenja. Cilj ovog rada je u sklopu suradnje s tvrtkom Hammer d.o.o. proučiti programsku podršku sustava za rukovanje portfeljima pod nazivom Insa te napisati i izvršiti skup automatiziranih testova u svrhu ispitivanja njene funkcionalnosti.

Drugo poglavlje daje informacije o metodologiji rada tvrtke Hammer d.o.o. uz pregled uobičajenih metodologija koje se primjenjuju u tvrtkama za izradu programskih rješenja. Poseban je naglasak stavljen na agilni model razvoja programske podrške pod nazivom scrum. Na kraju poglavlja obrazložen je razlog za provedbu automatiziranih testova te su opisane neke od tehnologija koje to omogućuju. Treće poglavlje posvećeno je opisu strukture aplikacije InsaWeb uz naglasak na tehnologije kojima je postignuto programsko rješenje. U sljedećem poglavlju obrađen je praktični dio rada koji se odnosi na pisanje automatiziranih testova u programskom jeziku C# pomoću alata Selenium i odgovarajućeg modela za organiziranje testova. Opisan je način rukovanja elementima internetskih stranica pomoću programskog koda te način postizanja ekonomičnosti programskog koda. Zadnjim poglavljem objašnjeno je povezivanje automatiziranih testova s uslugom na oblaku računala pomoću koje je moguće test izvršiti na višestrukim platformama. Prikazana je dokumentacija odabranih testova zajedno s njihovim rezultatima i ishodima u obliku tablica i slika. Svaki pojedini test je pojašnjen i prikladno komentiran.

2 POTREBE ZA AUTOMATIZIRANIM TESTIRANJEM PROGRAMSKE PODRŠKE

2.1 Metodologija tvrtke Hammer d.o.o.

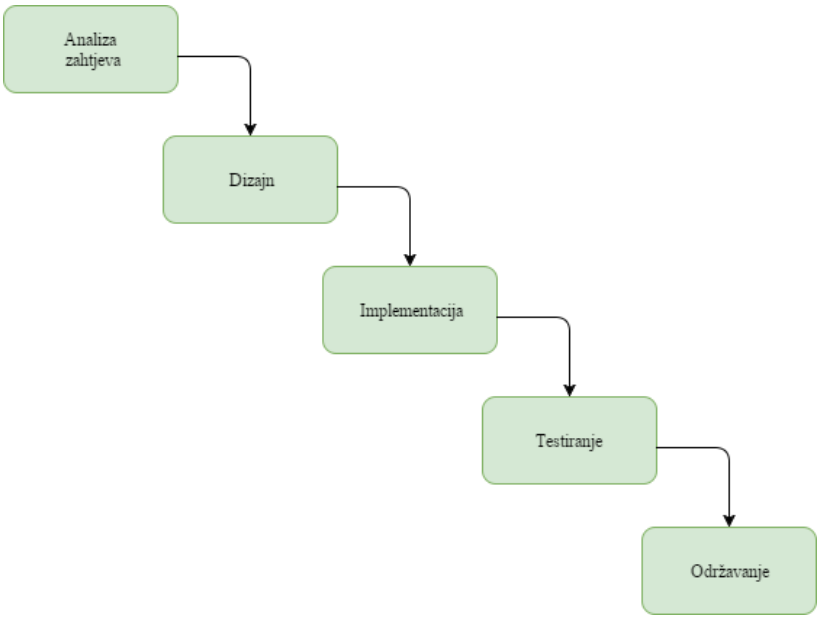
Insa je složeni sustav namijenjen za rukovanje portfeljima, generiranje financijskih izvještaja i provedbu analiza portfelja [1]. Imajući na umu izravnu povezanost ciljeva tvrtke te rastućih i promjenjivih trendova ekonomije, ključan segment poslovanja tvrtke je aktivna komunikacija i uspostavljanje odnosa s klijentima. Klijenti s kojima se vrši poslovanje su financijske organizacije poput banaka, osiguravajućih društava, mirovinskih fondova, posredništava i drugih fondova. Radi složenosti i jedinstvenosti financijskih sustava spomenutih organizacija, Insa je utemeljena na trenutnim praktičnim zahtjevima tvrtki klijenata, kao i budućim zahtjevima. Uzevši u obzir korisnički orijentiranu narav tvrtke i promjenjive zahtjeve i potrebe klijenata, od ključne je važnosti odabir odgovarajuće razvojne metodologije. Dvije su vodeće i najučestalije metodologije razvoja programske podrške, a to su metoda vodopada i skupina agilnih metodologija.

2.1.1 Metoda vodopada

Prema [2], metoda vodopada je poznata i pod nazivom linearni sekvencijalni model životnog ciklusa. Specifičnost ovog modela je način na koji se provode određene faze razvoja programske podrške. Prema metodi vodopada, svaka faza razvoja odvija se zasebno i u potpunosti neovisno o drugim fazama. Također, jedna faza može započeti s izvođenjem tek kada prethodna faza završi i ni u kojem slučaju ne smije doći do preklapanja istih. Slikovni prikaz modela vodopada vidljiv je na slici 2.1, načinjenoj prema [3].

Kao što je vidljivo sa slike 2.1, metoda vodopada se sastoji od pet ključnih faza, koje slijede jedna za drugom u procesu razvoja, bez vremenskog preklapanja. Prva faza ili analiza zahtjeva podrazumijeva detaljno istraživanje zahtjeva klijenata i sastavljanje popisa svih nužnih funkcionalnosti koje rješenje mora sadržavati, bez obzira radi li se o potrebama klijenta ili nužnih faktora bez kojih rješenje ne može biti potpuno. Arhitektura rješenja definira se u sljedećoj fazi. Proučavanjem zahtjeva koji su doneseni u prethodnoj fazi iznosi se dizajn cjelokupnog rješenja te zahtjevi sklopovlja. Kroz proces implementacije dizajn se provodi u djelo. Nakon implementacije rješenja, potrebno ga je testirati, budući da ga je

nemoguće sastaviti u fazi implementacije bez ijednog propusta ili greške. Testiranje predstavlja svojevrsnu barijeru koja sprječava konačni proizvod s određenim nedostatcima od izlaska na tržište. Također, u ovoj fazi se odrađuje i prepravljanje grešaka. Testiranje u pravilu traje sve dok se ne eliminiraju svi propusti. U zadnjoj fazi, rješenje se nalazi kod kupca, a proizvođač jamči njegovo održavanje i popravak u slučaju grešaka koje se mogu javiti u specifičnom klijentskom okruženju, a neotkrivene su prošle fazu testiranja. Uz to, proizvođač nudi nadogradnju isporučenog proizvoda.



Slika 2.1 Model vodopada

Prema [4], prednosti ovakve metodologije razvoja programske podrške leže upravo na strogom organizacijskom smještanju poslova u odvojene faze. U to se ubraja proces *departmentalizacije*, odnosno, podjele tvrtke i većeg posla na odjele i manje poslove. Također, olakšana je i kontrola nad procesom razvoja. Svaka se faza može odvojeno vrednovati kroz provjeru ostvarenih izlaznih vrijednosti, koje predstavljaju ulazne vrijednosti sljedeće faze, te kroz jasno definiranje rokova. Jednostavno je organizirati zadatke za pojedine faze jer ne zahtijeva razmišljanje o ostalim, potencijalno preklapajućim zadatcima. Rezultati pojedinog procesa su precizno i temeljito dokumentirani. Metoda vodopada je pogodna za manje, ne ponavljajuće projekte, gdje su zahtjevi također nepromjenjivi i jasno određeni. Također, iskustvo je pokazalo kako je metoda vodopada prirodno ugrađena u psihu razvojnog programera, kako je navedeno u [5].

Prema [2], nedostatci sekvencijalnog modela proizlaze iz istih karakteristika koje mogu biti izvor prednosti. S obzirom na strukturu odvojenih faza koje se ne preklapaju, iznimno je

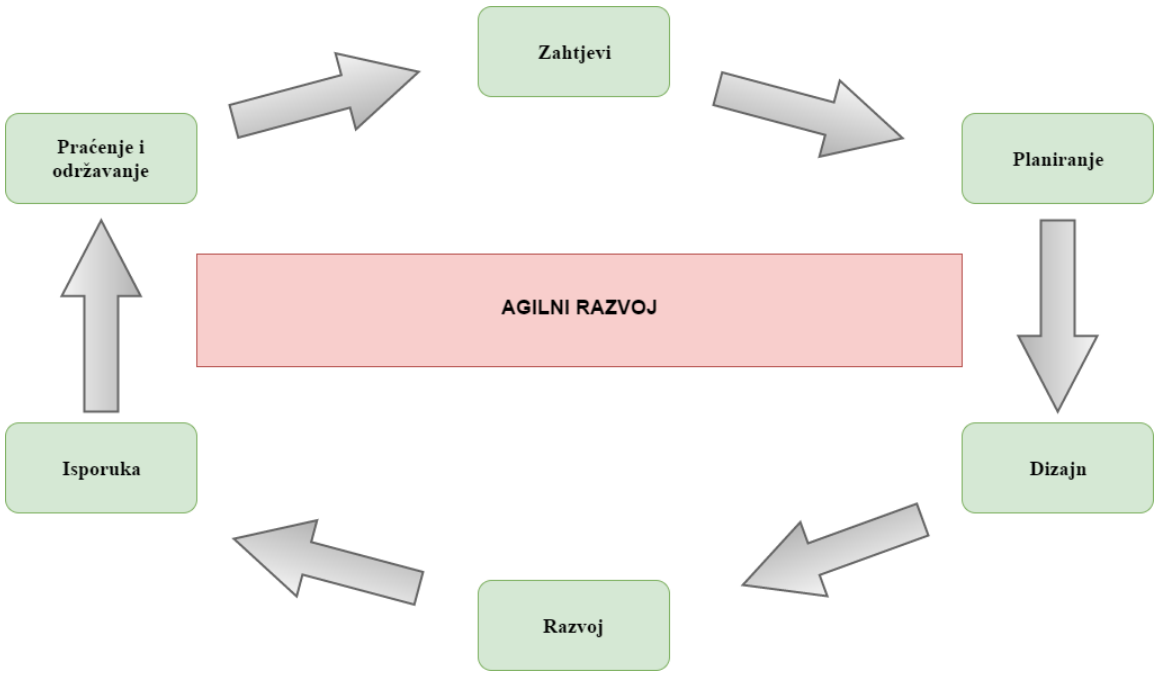
teško vratiti se na prethodnu fazu. Nije pogodan za projekte gdje se očekuje promjena zahtjeva od strane klijenta, stoga predstavlja visoko rizičan model. Rješenje s kojim je moguće raditi ne pojavljuje sve do kraja cjelokupnog procesa, odnosno, do zadnje faze, a to znači da će naručitelji dobiti gotov proizvod bez mogućnosti izmjene onoga što je već implementirano. Također, bilo kakva promjena smjera tijekom životnog ciklusa može značiti kraj projekta. Zbog navedenoga, model vodopada nije dobar model za složene i dugoročne projekte.

Uzevši u obzir prednosti i nedostatke ovog modela, vidljivo je kako on nije dobar izbor metodologije razvoja sustava Insa. Zbog korisnički orijentiranog poslovanja, složene strukture te dugoročnih ciljeva, metoda vodopada je neprikladna jer onemogućava fleksibilnost prema zahtjevima klijenata tijekom procesa implementacije, ispravke i dodatke nakon isporuke klijentima te neprekidnu nadogradnju i usklađivanje s potrebama tržišta. Klijenti često očekuju iskoristivost pojedine funkcionalnosti sustava, bez obzira na to je li objedinjena potpuna funkcionalnost, što ova metoda nije u mogućnosti pružiti jer naglašava čekanje gotovog proizvoda s razrađenom potpunom funkcionalnošću.

2.1.2 Agilne metodologije

Alternativa metodologiji vodopada su agilne metodologije. Agilni model omogućuje i pruža ono što model vodopada ne uspijeva, a to je brza isporuka dijelova rješenja koji su spremni za uporabu kroz ponavljajuće iteracije, od kojih svaka iteracija donosi potpuno funkcionalno programsko rješenje. Korisnički je usmjeren i potiče odnos između klijenta i razvojnog tima te usmjerava razvojni tim na zadovoljenje klijentskih potreba. Prema navodu iz [6], agilne metodologije ohrabruju formiranje malih razvojnih timova koji su visoko motivirani i jednostavnost procesa razvoja rješenja na širokoj razini. Agilnost podrazumijeva sposobnost brze prilagodbe na promjene, koje su karakteristične za razvoj programske podrške. Potreba za agilnošću pri razvoju programske podrške se javlja zbog rastuće tehnologije, koja omogućuje širok raspon odrađivanja istog cilja na više načina, pri čemu klijenti suočeni s velikim izborom često mijenjaju mišljenje. Također, određeni standardi tehnologije razvoja programske podrške koji se brzo mijenjaju u kratkom vremenu vrše pritisak na razvojni tim da njihovo rješenje bude u toku s brzim razvojem. Prema [6], nekoliko je karakteristika ključno za razvojni tim agilne metodologije. Neke od njih su kompetencija, odnosno, sposobnost izvršavanja procesa koji je trenutno na rasporedu, što uključuje i opseg znanja koji član tima mora posjedovati da bi mogao biti član agilnog tima. Ključan je zajednički

fokus koji moraju dijeliti svi članovi, a to podrazumijeva isporuku inkrementalnog rješenja koje je spremno za upotrebu u unaprijed dogovorenom vremenu. Kako bi se ostvarili svi zadaci koji su pred razvojnim timom, potrebna je suradnja između članova tima, suradnja tima i klijenta te tima i menadžera. Velika odgovornost leži na timu, jer posjeduju određen stupanj autonomije i zbog toga donose važne i velike odluke. Osim odluka, velika odgovornost je i pri rješavanju složenih problema, gdje je od visoke važnosti da odgovor na određeni problem ne uzrokuje buduće probleme. Zbog određene autonomije koju ima razvojni tim, samoorganizacija je od kritične važnosti, jer u izostanku iste može doći do kašnjenja isporuka iteracija, što uzrokuje sve veća kašnjenja u budućnosti. Na slici 2.2 prikazana je tipična iteracija, odnosno, ponavljajući životni ciklus agilnog razvoja, po uzoru na [7].



Slika 2.2 Model agilnog razvoja

Kao što prikazuje prethodna slika, teško je odrediti početnu fazu u razvoju, iako je nužno započeti od određene faze kada projekt kreće od nule. Faze razvoja su slične kao i faze pri modelu vodopada. U svakom projektu potrebno je znati koji su zahtjevi, odnosno, kakvu funkcionalnost je potrebno omogućiti u sklopu rješenja. Zatim slijedi planiranje strategije ili postavljanje pitanja „Kako to učiniti?“. U tu fazu ubrajaju se sklopovsko i programsko planiranje. Dizajniranje rješenja je korak u kojemu se razmatra konačan izgled rješenja. U tom dijelu razvoja događa se najdinamičniji posao, što podrazumijeva programiranje i testiranje onoga što je učinjeno. Nakon isporuke rješenja sa zadovoljavajućom

funkcionalnošću klijentu, slijedi proces komunikacije s klijentom u kojemu se saznaje o daljnjim greškama koje se javljaju isključivo u klijentskom okruženju, ali se nastavlja i osluškivanje zadovoljstva klijenta i njegovih budućih zahtjeva te potreba za unaprjeđivanjem isporučenog programskog rješenja. Iz te komunikacije ponovno se definiraju zahtjevi i ciklus se zatvara i ponovno pokreće. Važno je napomenuti kako niti jedna od ovih faza nije namijenjena samostalnom provođenju. Također, moguće je da se nekoliko faza izvodi istovremeno.

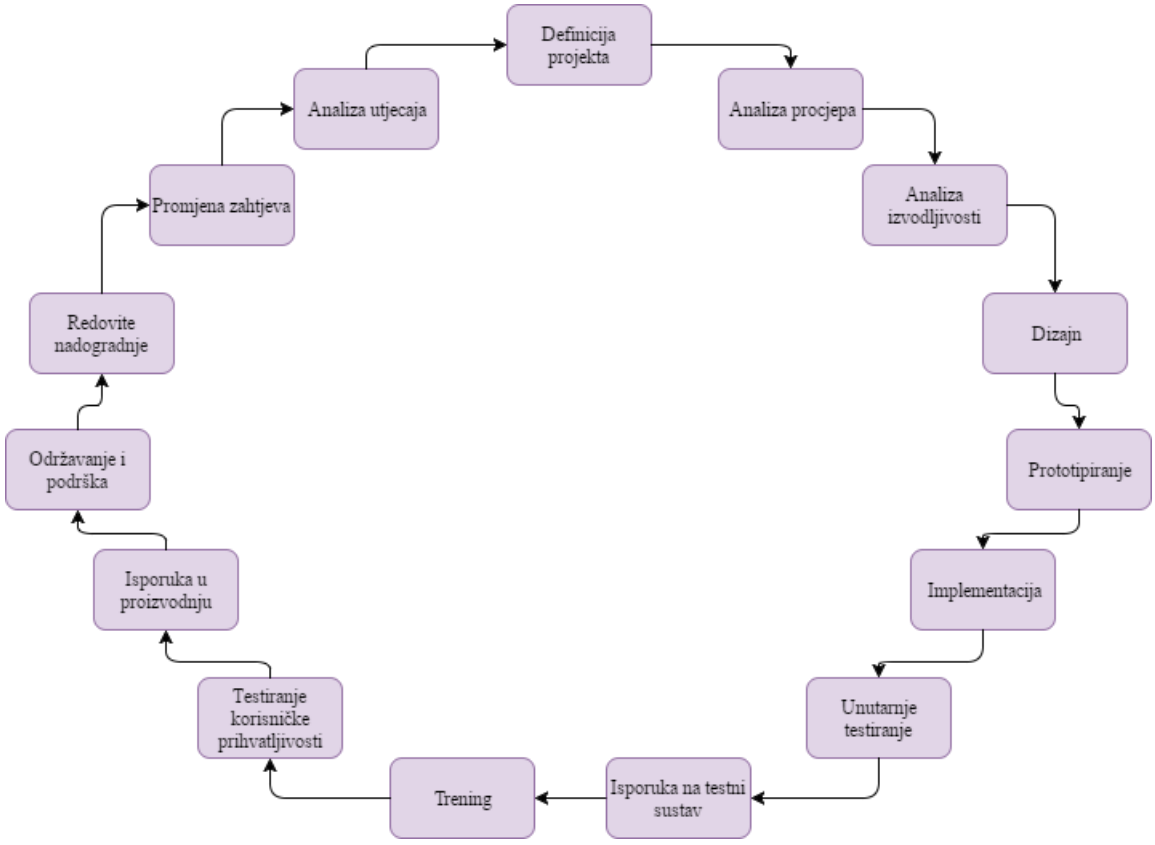
Implementacija agilne metodologije podrazumijeva redovito održavanje sastanaka, a učestalost njihovog održavanja varira ovisno o tome koliko dugo traje pojedina faza razvoja. Prema [8], sastanci trebaju biti kratki i sažeti te su izuzetno važni iz nekoliko razloga. Dugački sastanci su često zamorni za većinu članova razvojnog tima te uzimaju previše vremena. Ipak, za pojedinca je potrebno znati gdje se nalazi u odnosu na kolege te gdje se nalaze ostali pojedinci u odnosu na cjelokupan projekt. Također, ono što zna pojedinac u razvojnom timu trebali bi znati i ostali članovi tima. S druge strane, postoji opasnost u prečestom održavanju sastanaka, a ona leži u tome da klijent ima tendenciju preuzimati kontrolu nad projektom.

Slijede prednosti agilnih metodologija, opisanih prema [9]. Agilni pristup je realističan pristup izradi programskog rješenja, jer se usmjerava na korisničke zahtjeve i sustavnu komunikaciju s klijentom. Agilne metode jačaju tim, promiču timski rad i dijeljenje znanja i sposobnosti u svrhu postizanja istog cilja. Prikladne su za projekte gdje su zahtjevi poznati na početku i fiksni su tijekom procesa razvoja te za projekte čiji se zahtjevi mijenjaju tijekom vremena. Željena funkcionalnost se može brzo izraditi i predstaviti, bez nepotrebnog čekanja na potpuno gotov proizvod te s vrlo malo planiranja. Razvojni programeri su fleksibilni i mogu raditi istovremeno na više paralelnih faza razvoja.

Iako izgleda da je ovakav način organiziranja razvoja savršeno prikladan za svaki proces izrade programske podrške, postoji i nekoliko očitih nedostataka u takvom pogledu. Prema [9], agilne metodologije sa sobom nose visoku razinu rizika na području održavanja i proširenja. To je logično, s obzirom da razvojni tim ne zna s čim će se susresti u daljnjem tijeku agilnog ciklusa, odnosno, kakvi će zahtjevi i problemi proizaći kao rezultat suradnje s klijentom. Agilne metodologije podrazumijevaju uloge agilnog menadžera i projektnog vođu, što može biti problem za tvrtke koje nemaju zaposlenika takvog kalibra. Budući da se oslanjaju na interakciju s klijentima, može se dogoditi da klijenti u svojim potraživanjima ne budu dovoljno jasni i precizni, a razvojni tim može odvesti projekt u suprotnom smjeru od

zamišljenog. Također, novi članovi tima se zbog nedostatka detaljne dokumentacije teško snalaze u početku, što može usporiti cijeli tim.

Iako agilne metodologije posjeduju određene nedostatke s kojima se ne susreće metoda vodopada, zbog svojih glavnih prednosti i ključnih karakteristika je najbolji izbor za poslovanje razvojnog tima sustava Insa. Insa je usmjerena na trenutačne zahtjeve klijenata, a pokušava i pretpostaviti buduće te u tom smislu prati najnovije razvojne tehnologije i funkcionalnosti koje će u skoroj budućnosti biti od velike koristi klijentu. Zbog toga je potrebno primijeniti iterativne korake i uzastopce provjeravati sukladnost funkcionalnosti i vanjskih zahtjeva, kao i isporučivati dijelove rješenja s kojima se već može u potpunosti raditi. Kao sustav za olakšanje poslovanja u ekonomiji koji koriste različite financijske organizacije, od kritične je važnosti osluškivati pojedinog klijenta i približavati svoja rješenja njegovim specifičnim potrebama. Na slici 2.3 slijedi prikaz agilne metodologije razvoja prilagođen procesima u sustavu Insa.



Slika 2.3 Model agilne metodologije sustava Insa

Zbog visoke razine složenosti sustava Insa, njegov životni ciklus se sastoji od više koraka nego što se sastoji uobičajen prikaz modela agilnih metodologija. Međutim, bitno je napomenuti da niti jedan model nije univerzalan i nije namijenjen kao strogo ograničenje, već

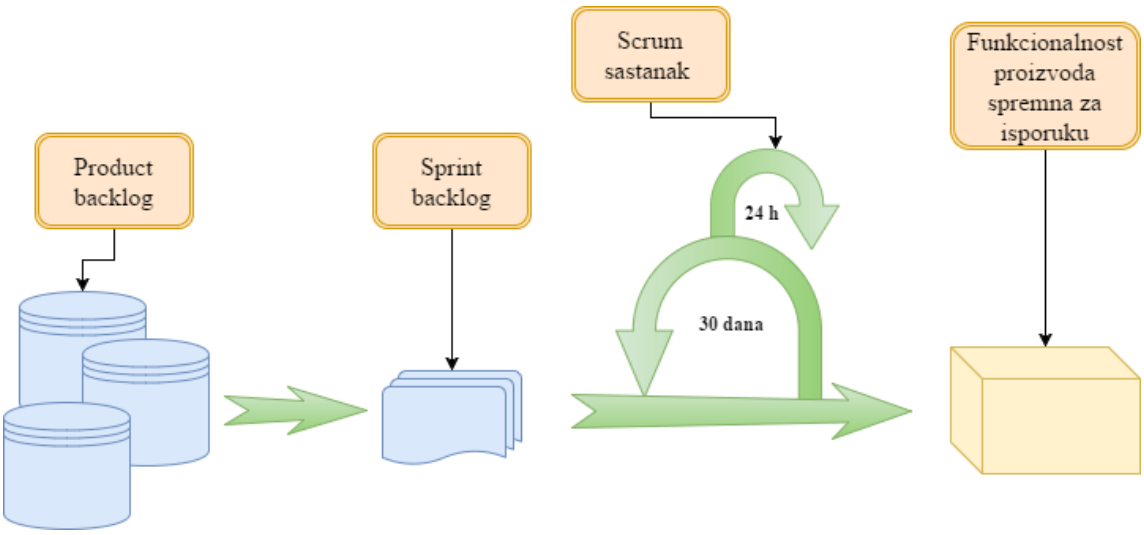
služi kao smjernica za modeliranje vlastitog životnog ciklusa projekta. Životni ciklus sustava Insa započinje definiranjem projekta, što vodi do usporedbe klijentskih zahtjeva i trenutačno raspoloživog rješenja. Nakon toga se provodi analiza stupnja izvodljivosti kojom se utvrđuje sposobnost udovoljavanja klijentskim zahtjevima. Nakon dizajniranja i izrade prototipa, odnosno, procesa izrade funkcionalnosti slijedi implementacija prototipa u sustav, gdje se provodi testiranje od strane razvojnog tima i testera. Zatim se isti proces ponavlja na testnom sustavu, gdje testiranje provodi korisnik. Nakon što proizvod prođe testove korisnika, slijedi puštanje rješenja u proizvodnju, no tu ciklus ne završava, nego se nastavlja kroz potpunu podršku i raspoloživost razvojnog tima u rješavanju problema i grešaka koje se mogu provući neopaženo u svim ranijim fazama testiranja. Uz pružanje usluge održavanja sustava, klijentima se jamči redovita nadogradnja sustava, čiji su razlog promjene u razvojnoj tehnologiji i držanje koraka s istima. Razvojni tim osluškuje potrebe klijenata i spreman je na njegovu promjenu mišljenja i pravovremeno reagira. Zbog promjena zahtjeva je potrebno izvršiti analizu koja će dati odgovor na pitanje kako će udovoljavanje izmijenjenim zahtjevima klijenta utjecati na cjelokupnu funkcionalnost sustava. Kada se ta faza završi, moguće je definirati novi projekt.

Iz spomenutog životnog ciklusa sustava Insa, vidljivo je kako se iste faze životnog ciklusa periodično pojavljuju u kraćim iteracijama. Budući da testiranje zauzima značajan dio ciklusa, potrebno je odvojiti velik dio vremena za njegovo provođenje. Uzevši u obzir da se potreba za testiranjem pojavljuje u svakoj iteraciji, za svaku pa i najmanju promjenu unutar rješenja, jasno je da svako ponovno provođenje i osmišljavanje testova oduzima dragocjeno vrijeme i resurse. Iz tog razloga je prikladno osmisliti skup testova koji će pokriti temeljne funkcionalnosti sustava kako bi se utvrdilo da neka nova nadogradnja nije poremetila rad prijašnjeg rješenja te koji će se provoditi bez potrebe za ljudskim navođenjem, odnosno, provodit će se automatski.

Iako životni ciklus razvoja sustava Insa predstavlja primjer primjene principa agilnih metodologija na razvoj programske podrške, potrebno je otići korak dublje kako bi se preciznije definirali pojedini koraci, njihovo vremensko ograničenje te komunikacija između članova razvojnog tima, projektnog menadžera te klijenata. Prema [6], modeli agilnih metodologija su ekstremno programiranje, adaptivni razvoj programske podrške (ASD), metoda razvoja dinamičkih sustava (DSDM), scrum, crystal, razvoj usmjeren svojstvima (FDD) te agilno modeliranje. Iako svaki pojedini od navedenih modela ima vlastite specifične karakteristike, svrha ovog pregleda nije njihovo definiranje. Daljnji opis će se usmjeriti na model scrum, iz razloga što je to model po kojemu se razvija sustav Insa.

2.1.3 Model Scrum

Kao što je navedeno u [6], scrum je agilni model razvoja programske podrške, koji svoj naziv duguje pojmu iz ragbija, gdje je cijeli tim uključen u pomicanju lopte prema naprijed. U suštini, upravo na taj način se u procesu razvoja programske podrške ponaša i razvojni tim. Naglasak scruma je na timskom radu, gdje se konačni rezultati zajednički iznose. Scrum potiče komunikaciju između članova tima, međusobnu suradnju te dijeljenje informacija i znanja. Rad na ukupnoj funkcionalnosti proizvoda je podijeljen na nisko ovisne, razdvojene segmente. Dok traje razvoj proizvoda, izvršavaju se redoviti testovi i dokumentiranje. Zbog neprestanih iteracija gdje se u svakoj od njih rješenje može testirati, nadograditi i prilagoditi, projekt se u bilo kojoj fazi može proglasiti dovršenim. Na slici 2.4, prema uzoru na [10] prikazan je tijek procesa modela scrum.



Slika 2.4 Tijek procesa modela scrum

Kao što je navedeno u [11], scrum proces se sastoji od odgovarajućih uloga, događaja i predmeta.

Postoje tri uloge unutar scrum procesa, a to su *scrum master*, vlasnik proizvoda i članovi tima. *Scrum master* je osoba koja je odgovorna za provođenje vrijednosti i principa scruma te otklanjanje ikakvih zapreka ili vanjskih utjecaja koji mogu negativno utjecati na tim i njegovu produktivnost. Njegova odgovornost leži i u osiguravanju produktivnosti i povezanosti suradnje tima i ostalih funkcija. Vlasnik proizvoda definira nove sastavnice proizvoda te odlučuje o rokovima isporuke istih. Na kraju svake iteracije ocjenjuje uspješnost rezultata, a njegova konačna odgovornost je osiguravanje profitabilnosti konačnog rješenja. Razvojni tim čini pet do devet članova, koji se međusobno raspodijeljeni po glavnim

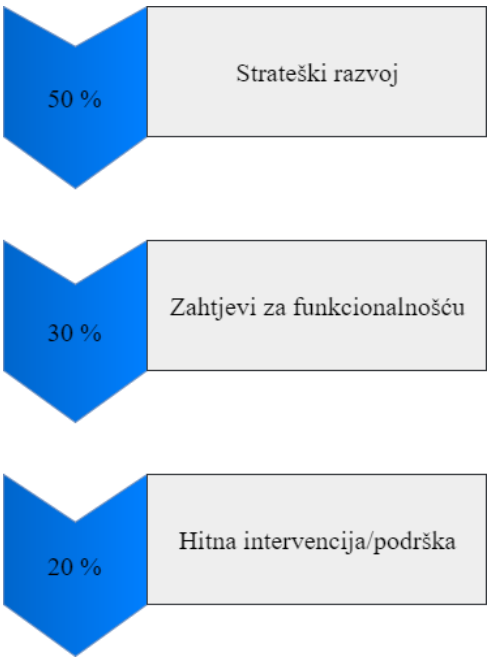
područjima struke, a to su programeri, testeri i dizajneri korisničkog sučelja. Na timu je odgovornost samostalne organizacije.

Članovi scrum projekta sudjeluju u scrum događajima. Scrum događaji su sprint, dnevni scrum sastanak, retrospektiva sprinta te osvrt na sprint. Sprint je pojam koji označava aktivnost u kojoj je glavni naglasak na razvijanju rješenja i testiranju. On traje od dva do četiri tjedna. Organizacija većeg posla u sprintove omogućuje bolju kontrolu nad projektom i smanjenje rizika. Dnevni scrum sastanak je sastanak u trajanju od petnaest minuta, iako može po potrebi trajati i duže. Scrum sastanak se odvija svakog dana i na njemu sudjeluju *scrum master* i članovi tima. Prema [6], na scrum sastanku svaki pojedini član tima odgovara na tri pitanja, a to su „Što sam radio jučer?“, „Što ću raditi danas?“, „Postoje li smetnje na mom putu?“. Održavanjem dnevnih scrum sastanaka svaki član tima, uključujući i *scrum mastera*, zna s kakvim se izazovima susreće njegov kolega i uskače u pomoć prema potrebi. Povremeno se organizira pregled dosadašnjeg učinka u sprintu pod nazivom retrospektiva sprinta. Na njima sudjeluju sve uloge u scrumu, a može uključivati i klijente. Na kraju svakog sprinta održava se prezentacija učinjenoga i demonstracija novih sastavnica ili funkcija. Takav sastanak je poznat pod nazivom osvrt na sprint.

Ključni predmeti unutar scruma su *product backlog* i *sprint backlog*. Pojam *product backlog* predstavlja popis zahtjeva za funkcionalnost i izgled konačnog proizvoda. Njegov sastav uređuje vlasnik proizvoda. Zahtjevi za određenim funkcionalnostima ulaze u scrum proces upravo kroz *product backlog*. Oni se dalje slažu u skupove zahtjeva koji su proslijeđeni u *sprint backlog*. Članovi tima biraju zadatke iz *sprint backloga* i odgovorni su za njihovo izvršenje u roku u kojem se izvršava trenutni sprint.

Scrum metoda je dovoljno fleksibilna da ju svaka tvrtka može primijeniti na svojoj specifičnoj infrastrukturi. Jednako tako i Insa posjeduje vlastitu inačicu scruma. Opis implementacije modela scrum u Insi je napisan po uzoru na službeni dokument Inse [1].

Sadržaj *product backloga* u Insi podijeljen je na tri glavna područja, kao što je prikazano na slici 2.5.



Slika 2.5 Sadržaj *product backloga* Inse

Strateški razvoj sadržava dugoročne ciljeve sustava i zadatke koji su usmjereni na njegovo unaprjeđenje i nadogradnju na način da klijenti u budućnosti budu zadovoljni. Zahtjevi za funkcionalnošću su zahtjevi klijenata koji podrazumijevaju promjenu postojećih funkcionalnosti ili dodatak nekih novih. Pravovremena podrška je ključna u odnosu s klijentima jer ona nudi rješenje u situacijama koje se mogu dogoditi isključivo u korisničkoj atmosferi.

Oko sustava Insa brinu se dva odvojena scrum tima, od kojih je jedan odgovoran za razvoj programske podrške, a drugi za implementaciju rješenja. Razvojni tim se sastoji od četiri razvojnih programera, jednog dizajnera korisničkog sučelja i jednog testera, iako testiranje provodi i svaki razvojni programer nakon što razvije svoj dio rješenja, neposredno prije nego ga preda testeru. Tim za implementaciju se sastoji od četiri upravitelja računa, osobe u korisničkoj podršci i agenta za prodaju.

Sprintovi u Insi su dvotjedni i na kraju svakog sprinta klijentima je dostupno razvojno rješenje koje mogu implementirati. Retrospektiva sprinta se odvija na kraju svakog sprinta i njome se prezentiraju sve izmjene i dodatci koje je donio prethodni sprint.

Osiguranje kvalitete

Jedna od glavnih karakteristika sustava Insa koja privlači i zadržava klijente je osiguranje kvalitete. Pogreška u programskom rješenju može uzrokovati lavinu pogrešnih podataka, koji

mogu izazvati kritičnu štetu u svijetu ekonomije i financija. Prema [1], osiguranje kvalitete se ostvaruje kroz pet stupnjeva testiranja, kao što je prikazano na slici 2.6.

Slika 2.6 Pet stupnjeva testiranja

1) Razvojni programer testira funkcionalnost koju je upravo razradio, a samo se testirana funkcionalnost prosljeđuje u privremenu inačicu rješenja.
2) Tester, koji je dio razvojnog tima testira svaku privremenu inačicu i aktivno komunicira s razvojnim programerima u svrhu otklanjanja grešaka.
3) Podrška prve razine provodi testiranje rješenja nakon testera. Vjerojatnost je da ona pronalazi greške koje nije pronašao niti jedan član razvojnog tima, jer ne komunicira s njima izravno, nego isključivo preko *scrum mastera*.
4) Upravitelj računa vrši posljednje testiranje rješenja neposredno prije same isporuke klijentu.
5) Zadaća klijenta je izvršavanje testova na vlastitom testnom sustavu. U tu svrhu izrađuje se kopija produkcijskih podataka nad kojima se izvršava testiranje, a nakon toga klijent ispunjava formular o korisničkom prihvaćanju rješenja. Moguće je da u ovoj fazi testiranja klijent otkrije greške koje su specifične za njegovo okruženje i postavke sustava.

Upravljanje inačicama

Upravljanje inačicama je pojam koji označava proces u kojemu se rješenje priprema za isporuku krajnjim korisnicima. Njegova glavna svrha je osigurati poštivanje rokova i pravovremenu isporuku. Prema [1], smjernice za upravljanje inačicama u Insi su kvartalne isporuke, dvotjedne inačice razvoja, razvojne inačice i inačice za isporuku u paralelnom izvođenju, potpuno testirane inačice za isporuku, izostanak novih sastavnica u inačici za isporuku te fiksna skupina sastavnica prije početka razvoja.

Insa pruža četiri nadogradnje sustava tijekom godine. Inačice za nadogradnju moraju biti potpuno testirane od strane članova tima. Dvotjedne inačice sustava su inačice koje klijenti mogu uključiti u konačno rješenje, ali preporučljivo je implementirati ih u testno okruženje. Takve inačice nisu dovoljno testirane kako bi mogle biti uključene u kvartalne nadogradnje sustava. Razvoj sustava je moguće izvoditi istovremeno s isporukom dijela rješenja jer se te

dvije radnje odvijaju na različitim granama procesa. Jedna od vodećih niti u isporuci inačice za nadogradnju je vođenje brige o potpuno testiranoj funkcionalnosti prije same isporuke.

2.2 Razlozi za automatizirano testiranje

Iz upravo opisanih principa osiguranja kvalitete i upravljanja inačicama sustava Insa, vidljivo je kako testiranje pojedinih inačica, bilo dvotjednih ili kvartalnih, ključan element koji daje sustavu vjerodostojnost i pouzdanost, što je i najvažnija karakteristika, s obzirom na područje interesa i ciljane klijente. Ako se uzme u obzir pet stupnjeva testiranja, kojima se ostvaruje osiguranje kvalitete, jasno je kako ručno testiranje na svakoj razini oduzima previše vremena, zbog čega dolazi do kašnjenja u isporuci dijela rješenja u trenutnom sprintu, a moguće je da se dogodi preljev kašnjenja na sljedeći sprint i na taj način prekrše rokovi kvartalne isporuke inačica za nadogradnju. Automatizirani testovi su rješenje koje se nameće u ovakvom sustavu i modelu razvoja sustava jer pružaju definiranje testova jednom, na početku procesa razvoja, a mogu se provesti koliko god je puta potrebno. Dobro utvrđeni automatizirani testovi mogu olakšati i ubrzati proces testiranja, koji se ciklički pojavljuje u višestrukim okolnostima paralelno s programskim razvojem i nakon programskog razvoja, sve do krajnje implementacije dijela rješenja u postojeći provjereni sustav. U tu svrhu, potrebno je izabrati optimalni alat za automatizirano testiranje, koji će biti primijenjen za definiranje i izvršavanje testova unutar razvojnog tima, prije isporuke programskog rješenja timu zaduženom za implementaciju.

2.3 Pregled postojećih tehnologija za automatizirano testiranje

Kada se govori o testiranju složene i opširne web aplikacije kao što je web inačica sustava Insa, koja je poznata pod nazivom InsaWeb, potrebno je imati u vidu da njezina složenost ne dolazi samo od njezine unutrašnje arhitekture ili načina na koji je implementirana određena tehnologija. Složenost je također prisutna i u pokušaju udovoljavanja velikom broju korisnika, od kojih svaki ima omiljeni web preglednik i operacijski sustav ili mobilni uređaj kojega se ne želi odreći kako bi se prilagodio aplikaciji. Naprotiv, korisnik očekuje da se aplikacija podredi njemu i pruži mogućnost odabira platforme na kojoj će pokrenuti InsaWeb. Budući da je upravo to jedan od glavnih svojstava aplikacije InsaWeb, potrebno je nakon svakog završenog dijela rješenja provesti testiranje na širokoj paleti uređaja, operacijskih sustava i web preglednika. Poprilično je skupo i nepraktično nabaviti velik broj mobilnih

uređaja, nakon svakog sprinta lokalno testirati aplikaciju na svakome od njih ili instalirati nekoliko inačica različitih web preglednika na različitim računalima te provesti ručno testiranje na njima. Iz toga razloga, donesen je zaključak da je potrebno pronaći uslugu na oblaku računala koja omogućuje upravo virtualizaciju svega navedenog, kojom će biti moguće ispitati sve potrebne kombinacije tehnologija kojom se može služiti korisnik aplikacije te na svakoj pokrenuti unaprijed definirani skup testova, kako bi se ubrzao sam proces testiranja. Ukratko, slijedi pregled nekih od najzastupljenijih usluga za automatizirano testiranje web aplikacija. Najviše je pažnje ipak posvećeno Browserstacku, budući da je praktični dio rada odrađen na njemu.

2.3.1 Saucelabs

Saucelabs je mrežna platforma namijenjena za automatizirano testiranje [12]. Njezin je cilj ubrzanje procesa testiranja te podizanje kapaciteta testova. Omogućuje testiranje prema unaprijed definiranim scenarijima. Paralelnim testovima pruža mogućnost obavljanja posla u kratkom vremenskom roku, što znači da istovremeno može pokrenuti više testova, bilo da se isti test odvija na različitim kombinacijama uređaja i web preglednika ili da se odvija više različitih testova. Pri svakom testiranju korisniku daje uvid u upravo testirani scenarij kroz snimljeni video zapis cijelog testa ili kroz skup snimki zaslona. Ono što je bitno za svakog testera jest široki pojas podržanosti tehnologija za pisanje testova. Testovi koji se izvršavaju na Saucelabsu mogu biti pisani u bilo kojem programskom jeziku i bilo kojem programskom okviru. Također, podržano je testiranje uživo, odnosno, upravljanje udaljenim virtualnim strojem izvan okvira automatiziranih testova, a moguće je i ubaciti se u automatizirani test te preuzeti kontrolu od određene točke na dalje. U svrhu pokretanja testova na različitim kombinacijama operacijskih sustava i web preglednika, Saucelabs nudi preko 800 takvih kombinacija te 140 emulatora i simulatora mobilnih uređaja. Među njima su, dakako, Android i iOS. Dodatna mogućnost pri testiranju na mobilnim uređajima jest iznajmljivanje privatnog oblaka uređaja. U njemu korisnik može pristupiti stvarnim mobilnim uređajima koji su prilagođeni njegovim zahtjevima i čija konfiguracija odgovara u potpunosti njegovim potrebama te nitko drugi nema pristup prema njima niti može pokretati testove na njima.

2.3.2 Browsera

Još jedan u nizu alata za automatizirano testiranje web aplikacija u višestrukim kombinacijama operacijskih sustava i preglednika je Browsera. Njezin princip rada je

drugačiji od onog koji podržava Saucelabs, stoga je vrijedno proučiti je. Prema [13], slijede glavne karakteristike i mogućnosti koje nudi Browsera. Browsera je zamišljena da oponaša akcije testera. Pod tim se ponajprije podrazumijeva vizualno opažanje razlika na web stranici pristupljenoj preko različitih web preglednika. S obzirom na količinu vremena koja je potrebna da tester usporedi izgled stranice na više različitih preglednika te sposobnost detektiranja svih postojećih razlika, računalna moć donosi bitno poboljšanje u takvom vidu testiranja. Browsera uzima snimke ekrana na svakom pregledniku te ih uspoređuje i obavještava korisnika o potencijalnim problemima. Ukoliko se pri učitavanju stranice ili određenog elementa stranice otkrije greška učinjena u JavaScript kodu, detalji greške su dostupni korisniku u izvještaju o izvršenom testu. Umjesto testiranja jedne pod stranice za drugom pritiskom na poveznice prema njima, Browsera pruža mogućnost testiranja svake podstranice istovremeno, što dodatno smanjuje vrijeme izvršenja testa. Također, u slučaju susreta s dinamičkim sadržajem, pričekat će s učitavanjem svakog elementa prije nego uzme snimku zaslona. Za razliku od ostalih sličnih alata na oblaku računala za testiranje, Browsera vrši testiranje sam, na osnovu prethodno definiranih početnih postavki. S druge strane, brzina izvršavanja testa koja dolazi s tom karakteristikom donosi i smanjenu kontrolu nad izvršavanjem testa, odnosno, nad preciznim definiranjem svakog pojedinog testa. Zbog toga se ubraja među komparativne alate za testiranje, za razliku od analitičkih alata poput Browserstacka.

2.3.3 Browserstack

Browserstack je usluga na oblaku računala koja omogućuje testiranje funkcionalnosti i dizajna web aplikacija u višestrukim scenarijima koji podrazumijevaju različite operacijske sustave te web preglednike [14]. Također, u kontekst simuliranih korisničkih scenarija ubraja se i raspon različitih elektroničkih uređaja, od osobnih računala do pametnih telefona. Zbog široke palete mogućnosti koju pruža uspostavljanjem virtualnog stroja za svaku odabranu kombinaciju operativnog sustava i web preglednika te korisnički usmjerenog dizajna i intuitivnih kontrola, vrlo je lako koristiti neke od osnovnih segmenata koje nudi. Dva su glavna načina testiranja aplikacija koje nudi Browserstack, a to su testiranje uživo (*Live*) i automatizirano testiranje (*Automate)*. U nastavku slijedi opis oba načina testiranja.

Mogućnost testiranja uživo omogućuje korisniku odabir uređaja na kojemu je potrebno izvršiti test. Glavna podjela se sastoji od osobnih računala i mobilnih uređaja. Osobna se računala dijele na računala operacijskog sustava Windows i računala Mac. Browserstack nudi

raspon različitih distribucija Windows operacijskog sustava, od Windows XP do Windows 10. Također, ubrojene su i različite distribucije Macintosh, od OS X Snow Leopard do macOS Sierra. Web preglednici koje je moguće odabrati su Internet Explorer, Microsoft Edge, Mozzila Firefox, Google Chrome, Opera, Yandex te Safari. Dodatno, svaki web preglednik se grana na nekoliko prijašnjih inačica. Vidljivo je iz spomenutog kako se od korisnika traži da bude vrlo specifičan pri odabiru ciljane platforme na kojoj će se izvršiti testiranje, što je vrlo pogodno u odnosu na zahtjeve kupaca web aplikacija, koji se neće odreći omiljenog ili po navici ustaljenog operativnog sustava i web preglednika, bez obzira ima li razvojni programer ili tester pristup pravoj kombinaciji specifikacija koja odgovara kombinaciji koju oni koriste. Potpuni popis uređaja i web preglednika čiju virtualizaciju nudi Browserstack nalazi se na službenim stranicama usluge [15]. Ono što čini testiranje uživo putem Browserstacka moćnim i vjerodostojnim jest virtualizacija stvarnih web preglednika na stvarnim uređajima sa svim popratnim mogućnostima koje podržava svaki web preglednik instaliran na lokalnom uređaju kojeg korisnik fizički posjeduje. Te mogućnosti podrazumijevaju alate za razvojne programere koji omogućava detektiranje grešaka i navigaciju kroz elemente web stranice, prečace na tipkovnici, kopiranje i lijepljenje teksta, pregledavanje video zapisa, preslušavanje audio zapisa, praćenje prijenosa slike i zvuka uživo te unaprijed instalirane dodatke poput Flash Player, Java, Silverlight, Acrobat Reader i drugih, koje lokalno instalirani web preglednici ne sadrže neposredno nakon instalacije na fizičkom uređaju.

Automatizirano testiranje omogućuje pisanje scenarija za određene testne slučajeve, koji će se na zahtjev testera pokrenuti i razriješiti ga napora ručnog prolaska kroz aplikaciju. Takvi testovi su sposobni, prema ranije definiranim koracima u scenariju testa, izvršiti sve što izvršava i čovjek korištenjem miša i tipkovnice, s naglaskom na brzinu i efikasnost prolaska kroz test. Automatizirano testiranje olakšava posao u smislu da pruža testeru potpuni oslonac i razrješava ga ponavljajućih, monotonih i dosadnih radnji kao što su pokreti mišem i tipkovnicom te omogućava prolazak kroz scenarij testa znatno brže nego što je čovjek za to sposoban. Za vrijeme odrađivanja scenarija, tester ima vremena raditi na nekom drugom testu ili projektu ili pripremati izvješće. S druge strane, odgovornost na testeru je napisati kvalitetan scenarij u određenom programskom ili skriptnom jeziku, tako da scenarij ne naiđe na pogreške tijekom svog izvršavanja i ne dovrši scenarij testa. Takva je odgovornost veća od odgovornosti pri ručnom testiranju uživo. Browserstack omogućuje istovremeno pokretanje i izvršavanje testova na više različitih preglednika, što dodatno skraćuje vrijeme potrebno za testiranje aplikacije. Scenariji mogu biti pisani u raznim popularnim programskim jezicima,

kao što su C#, Java, Ruby, Python i drugi. Postoje dva alata kojima se testovi mogu automatizirati na Browserstacku, a to su Selenium i JavaScript. Praktični dio koji će biti kasnije opisan, odrađen je pomoću Seleniuma, stoga je naglasak stavljen na taj skupu alata za automatizirano testiranje. Kao i u slučaju testiranja uživo na Browserstacku, u slučaju automatiziranog testiranja je moguće odabrati operacijski sustav, web preglednik te rezoluciju ekrana stvarne platforme za koju je namijenjena aplikacija koja se testira. Jedina je razlika u tome što se željene postavke definiraju u programskom kodu, za razliku od interaktivnog odabira kroz sučelje, što je slučaj kod testiranja uživo.

Napisani testovi se mogu organizirati u skupine pod nazivom *build,* a *buildovi* se mogu organizirati u projekte. U skupinu *build* se organiziraju uzastopni pokrenuti testovi, najčešće vremenski povezani, pokrenuti istog dana ili kraćeg vremenskog perioda. U skupinu projekata se organiziraju različite inačice istog testa koje su tematski vezane. Moguće je onemogućiti i Flash dodatke na pregledniku, kao i onemogućiti skočne prozore, koji mogu svojim pojavljivanjem zakloniti neke kontrole, gumbe ili poveznice na internetskoj stranici na kojoj se izvršava test. U svrhu pronalaženja grešaka na testovima koji se ne uspiju završiti, Browserstack posjeduje zapise koraka kroz koje je test prošao te njihovu prikladnu oznaku koja govori o uspješnosti izvršavanja koraka. Raspoloživ je i zapis iz konzole web preglednika na kojem se izvršava test, a posebno koristan je i video zapis na kojem tester može vidjeti kako je odrađen svaki pojedini korak u testu. Ukoliko je testeru zanimljiv samo određeni korak testa i ne želi pretraživati video zapis, dostupan je i trenutni prikaz ekrana, koji je nakon odrađene akcije spremljen kao slikovni zapis.

2.3.4 Selenium

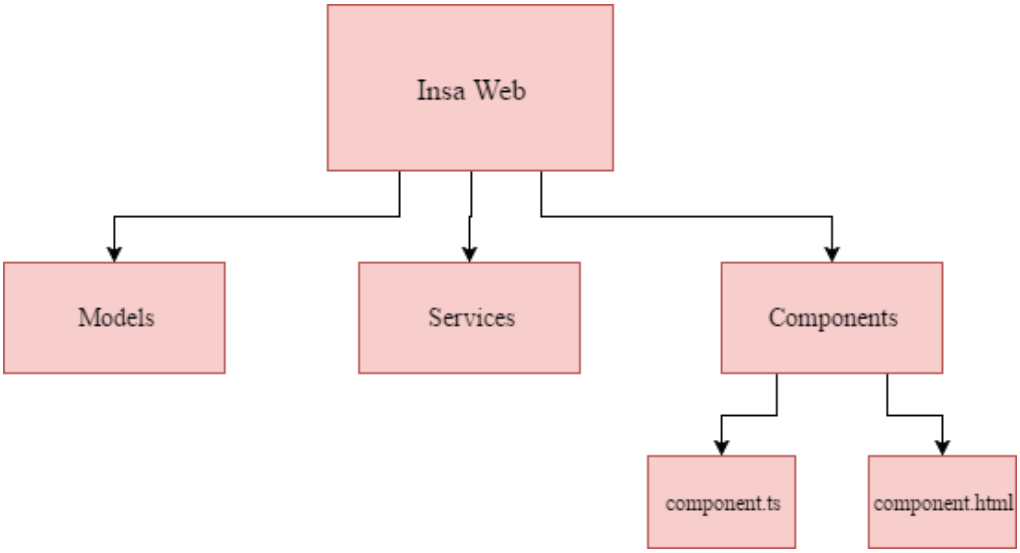
Selenium je skup alata koji pružaju podršku za automatizirano testiranje [16]. On sadrži višestruke funkcije, koje su vrlo rastezljive i prilagodljive, a služe prepoznavanju elemenata korisničkog sučelja i interakciji s njima. Zbog raznovrsnosti alata, njegova uporaba seže i dalje od samog testiranja te je prikladan za automatiziranje bilo kakvog unaprijed poznatog ponavljajućeg skupa naredbi na internetskoj stranici, čiju svrhu određuje korisnik. Selenium je započet kao JavaScript biblioteka koja je omogućavala interakciju s internetskim stranicama, a s vremenom se biblioteka razvila na način da je omogućila kontrolu nad preglednikom pomoću višestrukih programskih jezika. S porastom uporabe web tehnologija i napretkom na tom polju, Selenium se povezao s projektom WebDriver, koji nadopunjuje biblioteku Selenium s načinima dohvaćanja promjenjivih elemenata na internetskoj stranici,

dok se sama stranica ne mijenja. Suradnja ta dva projekta je nazvan Selenium 2.0 ili intuitivnije, Selenium WebDriver, a upravo je ta inačica korištena prilikom izrade praktičnog dijela ovog rada. WebDriver pruža API koji olakšava upravljanje dinamičnim elementima stranice, na način da ih promatra kao objekte. Za različite web preglednike postoji zasebna inačica WebDrivera. Selenium 2.0 obavlja izravne pozive prema pregledniku koristeći ugrađenu potporu preglednika za automatiziranje. Način na koji se obavlja poziv prema pregledniku je jedinstven za svaki preglednik. Iz tog razloga je nužno postojanje različitih inačica WebDrivera, a one su Firefox Driver, Internet Explorer Driver, Chrome Driver te Opera Driver. Posebnu skupinu čine iOS Driver te Android Driver, koji su namijenjeni automatiziranju naredbi na mobilnim uređajima.

Mogućnosti koje pruža Selenium 2.0 u smislu kontrole na stranicom i njezinim elementima su brojne i raznolike te se neprestano razvijaju u skladu s web tehnologijama, kako bi omogućili radnje koje su omogućene i korisniku koji ručno istražuje stranicu. Osnovna mogućnost je dohvaćanje same stranice te dohvaćanje elemenata stranice preko različitih identifikatora, koji mogu biti ime elementa, ID, ime pripadajuće klase, ime oznake, XPath i drugi. Izbor identifikatora elemenata stranice je specifičan, to jest ovisan o ciljanom rezultatu korisnika i strukturi internetske stranice. Nadalje, WebDriver omogućuje dohvaćanje tekstualnih zapisa na stranici, popunjavanje forme, kretanje kroz više istovremeno otvorenih prozora, upravljanje skočnim prozorima poput prozora upozorenja ili prozora obavijesti te navigaciju kroz povijest pretraživanja.

3 MODEL TESTNE WEB APLIKACIJE

Insa je moderan i složen sustav za rukovanje portfeljima i praćenje financijskih transakcija. Insa je izvorno zamišljena kao aplikacija za osobna računala, međutim, zbog najnovijih rastućih trendova uporabe mobilne i mrežne tehnologije, zahtjevi klijenata pokrenuli su se u smjeru prijenosnih i prilagodljivih platformi uz zadržanu izvornu funkcionalnost aplikacije. S trendom prijenosa složenih aplikacija u mrežni i mobilni svijet, javlja se i problem usklađivanja tehnologije za izradu aplikacija s širokom paletom mobilnih uređaja različitih proizvođača. Ako se uzme u obzir raznolikost operacijskih sustava koji pokreću mobilne uređaje i osobna računala, brojnost internetskih preglednika i njihovih inačica te rasprostranjenost rezolucija ekrana od onih na ekranima osobnih računala do rezolucija mobilnih uređaja, jasno je kako prijenosna inačica Inse mora udovoljiti mnogim zahtjevima i izazovima. Razvojem web inačice Inse, glavni cilj je postići univerzalnost, odnosno, potpunu funkcionalnost programske logike neovisno o operacijskom sustavu i pregledniku na kojima će se pokrenuti. Drugi, jednako važan cilj je premostiti razlike u rezoluciji ekrana uređaja preko kojih će korisnici pristupiti aplikaciji implementacijom responzivnog dizajna. Ako se navedeni izazovi uzmu u obzir, jasno je kako će bez obzira na stručnost razvojnih programera i dizajnera doći do neusklađenosti, odnosno, pogrešaka u izradi rješenja. Upravo zbog toga do naglaska dolazi funkcija testiranja, koja obuhvaća ispitivanje ne samo funkcionalnosti aplikacije, nego i njene skalabilnosti, to jest provjeru hoće li se iste kontrole, funkcije i elementi korisničkog sučelja jednako ponašati u različitim okruženjima i uvjetima. U svrhu razumijevanja strukture aplikacije na kojoj će se izvršiti testovi, slijedi opis tehnologija korištenih za njezinu izradu. Izrada *frontenda* odvija se unutar razvojnog okvira Angular 2, komunikacija s *backendom* je odrađena pomoću Web API-ja, a uzorak po kojemu je definiran arhitekturni dizajn je MVVM, prikladan zbog unutarnje arhitekture Angular 2. Na slici 3.1 prikazana je struktura aplikacije InsaWeb, sa svim pripadajućim sastavnicama.

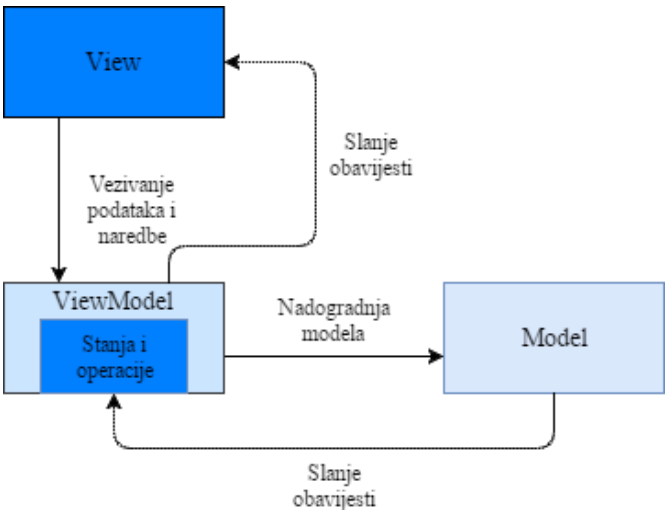


Slika 3.1 Struktura aplikacije InsaWeb

3.1 Uzorak MVVM

Prema [17], MVVM (Model – View – ViewModel) je uzorak po kojemu se definira arhitekturni dizajn programskog rješenja. Dio je skupine uzoraka za izradu programskih rješenja zajedno s MVC i MVP uzorkom. Potreba za uvođenjem uzoraka u razvoj korisnički orijentiranih aplikacija javila se porastom njihove složenosti i opsega. Također, korištenje uzoraka u izradi aplikacija postaje osnovna potreba uvođenjem agilnih metodologija, gdje je svaki član razvojnog tima zadužen za pojedinu funkcionalnost rješenja, a uloga razvojnog programera u potpunosti razdvojena od uloge dizajnera korisničkog sučelja i testera. Uzevši to u obzir, jasno je kako različite uloge u razvojnom timu moraju imati zagarantiranu mogućnost istovremenog rada na istoj aplikaciji, neovisno jedan o drugome. MVVM u tu svrhu pruža jasnu razdvojenost između programerske logike i korisničkog dizajna, kao što je navedeno u [18], što omogućuje istovremeni rad razvojnog programera i dizajnera na istoj aplikaciji. Nadalje, razdvojeni koncepti programske logike i korisničkog sučelja olakšavaju proces testiranja. Tester može provjeriti ispravnost dizajna neovisno o programskoj logici i obrnuto. Ta je osobina posebno korisna u kasnijem procesu ispravljanja pogrešaka jer pogreške koje se tiču dizajna pronađene prilikom testiranja se mogu ispraviti na način da ispravke ne utječu na programersku logiku, koja je sadržana u pozadini aplikacije. Primjerice, ako dizajner iz nekog razloga odluči promijeniti izgled korisničkog sučelja, može to učiniti na način da razvojni programer ne mora mijenjati i prilagođavati svoj dio rješenja. S porastom složenosti korisnički usmjerenih aplikacija, ekonomičnost koda, odnosno, ponovna upotrebljivost segmenata programskog koda postaje jedan od osnovnih zahtjeva. MVVM

upravo omogućuje ponovljenu iskoristivost dijelova rješenja unutar projekta. Kao što i samo ime govori, MVVM uzorak se sastoji od tri dijela, a izgled uzorka vidljiv je na slici 3.2, izrađenoj prema [19].



Slika 3.2 Uzorak MVVM

Model

Model predstavlja stvarne podatke nad kojima se vrše operacije dohvaćanja i uređivanja. Model sadržava podatke, ali ne i opise ponašanja podataka. Takva organizacija podataka je bliska fizičkom uređenju spremnika resursa, koji služi samo pohranjivanju resursa, dok ostali alati i procesi rukuju resursima prema potrebi.

View

View je, zapravo, izgled koji korisnik dobiva na raspolaganje pri korištenju aplikacijom. On je korisničko sučelje koje omogućuje korisniku rukovanje svim omogućenim funkcionalnostima programskog rješenja. Nije namijenjen definiranju odnosa između podataka niti njihovom dohvaćanju iz Modela, nego prezentaciji organiziranih podataka koje dohvaća iz ViewModela putem procesa vezivanja podataka. Također, dvostranu komunikaciju s ViewModelom održava podizanjem događaja, na koje ViewModel s druge strane reagira i vrši predviđenu logiku rada.

ViewModel

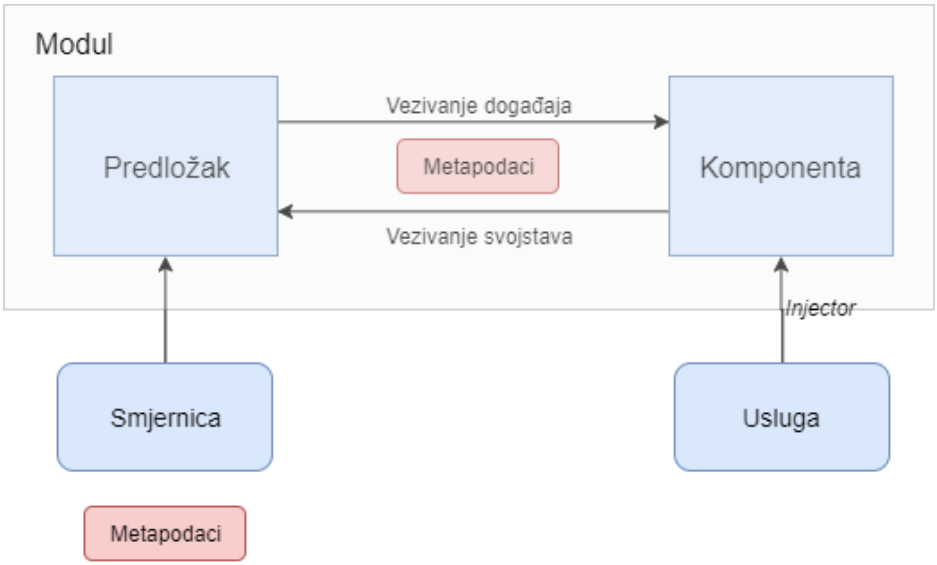
Ključni dio uzorka MVVM je upravo ViewModel, koji ovaj uzorak razlikuje od ostala dva. ViewModel definira metode i naredbe koje rade s podacima smještenima u Modelu, a koje su

pozvane uslijed korisničkih interakcija s Viewom. On osigurava potpunu neovisnost između Modela i Viewa, na način da im omogućuje posredničku komunikaciju. Cijela programerska logika sadržana je u njemu, što podrazumijeva pohranu novih podataka u Model, čitanje podataka iz Modela, organiziranje i prosljeđivanje istih prema Viewu te definiranje svojstava podataka, odnosno, informacija o podacima.

MVVM nije razvojni okvir, nego samo uzorak koji se može iskoristiti u svrhu izrade arhitekture aplikacije, iako ne pruža cjelovito rješenje za aplikacijsku strukturu. On ne vodi brigu o tome što se događa na poslužitelju, niti kako usluge međusobno djeluju, kao što je navedeno u [18]. MVVM je namijenjen ubrzanju procesa razvoja korisnički orijentiranih aplikacija, omogućujući pisanje dijelova programskog koda koji se mogu ponovno koristiti, odvojeno razvijanje programske logike, dizajniranje izgleda korisničkog sučelja i testiranje te arhitekturu prikladnu za koncepte agilnih metodologija razvoja programskog rješenja.

3.2 Razvojni okvir Angular

Angular je nastao kao projekt čiji je cilj bio olakšati razvojnim programerima i dizajnerima razvoj web aplikacija korištenjem HTML oznaka. Naziv Angular upravo dolazi od engleske riječi za vrstu uglatih zagrada koje se koriste za pisanje HTML oznaka i odnosi se na razvojni okvir koji koristi skriptni jezik JavaScript, a omogućuje izradu modernih web aplikacija kompleksnih zahtjeva [20]. Njegova biblioteka pruža mogućnost relativno lake primjene vezivanja podataka, usmjeravanja i izrade animacija. Angular je najprikladniji za izradu aplikacija srednje do visoke složenosti na kojima radi nekoliko članova tima. Također, prikladan je i za aplikacije koje su namijenjene izvođenju na različitim razvojnim okruženjima ili platformama. Primjerice, Angular omogućuje izradu aplikacije koja se može izvršavati na web pregledniku na operacijskim sustavima Windows i OSX te na mobilnim uređajima operacijskih sustava Android i iOS. Upravo zbog tog svojstva je upotrjebljen za izradu aplikacije InsaWeb, čiji ciljani kupci očekuju funkcionalnost izvođenja na različitim uređajima i platformama. Trenutno su dostupne dvije inačice Angulara, Angular 1, što je prva inačica biblioteke te novija inačica pod nazivom Angular, koja je u suštini potpuno prepisana prva inačica namijenjena novijim preglednicima i razvojnim platformama. Sastavnice arhitekture Angulara prikazane su na slici 3.3, a nakon slike slijedi opis sastavnica, nabrojanih prema [21].



Slika 3.3 Arhitektura Angulara

Modul je sastavni dio Angular aplikacije. Svaka aplikacija sastoji se od jednog osnovnog i više dodatnih modula. Modul se sastoji od dijela programskog koda koji se može iskoristiti za izvođenje pojedinog zadatka. Dijelovi modula se mogu izvoziti, a najčešće se izvozi klasa pod nazivom komponenta.

Komponenta je kontrolirajuća klasa čija je uloga kreiranje izgleda aplikacije, razmještaja elemenata na stranici i definiranje njihovih međuodnosa. Komponentu čini dio programskog koda koji se može ponovno koristiti unutar aplikacije. Izgled komponente može se definirati korištenjem predložaka, koji govore na koji način će Angular prikazati komponentu.

Metapodaci zajedno s komponentom i predloškom čine pogled aplikacije. Komponenta bez pripadajućeg metapodatka je samo klasa. Ona postaje komponenta tek kada joj se pridruži metapodatak koji Angularu daje do znanja da se radi o komponenti, a ne o običnoj klasi.

Vezivanje podataka je proces u kojem se segmenti predloška povezuju sa segmentima komponente. Njime se definira veza između izvora podataka i njihovog odredišta, koji su HTML elementi. Prema [21], postoje četiri vrste vezivanja podataka. Interpolacija je vezivanje koje prikazuje vrijednost komponente unutar div oznaka. Vezivanje svojstava služi prenošenju svojstava s komponente roditelja na komponentu dijete. Vezivanje događaja omogućuje pozivanje određene metode kao reakciju na korisničke akcije poput klika mišem. Dvosmjerno vezivanje omogućuje istovremeno vezivanje svojstava i događaja.

Usluge su funkcije jezika JavaScript koje pružaju mogućnost provedbe određenog posla. Usluge u Angularu se ubacuju putem mehanizma pod imenom *dependency injection*. Usluge su klase koje komponenti mogu omogućiti određenu vrijednost, metodu ili dodatke, prema zahtjevima aplikacije.

Smjernica je klasa koja predstavlja metapodatke. Postoje tri smjernice u Angularu. Prva je strukturalna smjernica, koja razmješta raspored elemenata u pogledu. Smjernica atributa mijenja izgled i ponašanje HTML elemenata. Zadnja smjernica je smjernica komponente, koja stvara vlastiti upravljač, koji se koristi kao zasebni HTML element.

Dependency injection je način prosljeđivanja ovisnosti klasi, koje se odvija prilikom stvaranja svake njezine nove instance, čime se postiže ekonomičnost koda i smanjenje ovisnosti pojedinih komponenti [22]. U Angularu većina ovisnosti su usluge, koji se upravo putem *dependency injectiona* prosljeđuju novim komponentama. Pri stvaranju komponente, zadaća *injectora* je da potraži odgovarajuću uslugu koji je potreban komponenti.

3.3 Uzorak REST

Skraćenica REST označava „Representational State Transfer“ i po definiciji [23] je arhitekturni uzorak za izradu raspodijeljenih hipermedijskih sustava. Pojam REST prvi je uveo Roy Fielding u svojoj doktorskoj disertaciji iz 2000. godine. REST koristi HTTP protokol, što je osnovni internetski protokol koji omogućuje komunikaciju između klijenta i krajnje točke, gdje klijent i krajnja točka mogu biti bilo što. Tipičan primjer bila bi komunikacija između web preglednika, koji predstavlja klijenta i poslužitelja kao krajnje točke. Prema [23], REST je definiran kroz šest ograničenja:

1) Klijent – poslužitelj

   Poslužiteljska strana se ne bavi problemima korisničkog sučelja, dok klijentsku stranu ne zanima način pohrane podataka. Na taj način korisničko sučelje se odlikuje visokom razinom portabilnosti, a poslužiteljske sastavnice postaju manje složene, što pridonosi skalabilnosti.

2) Bez stanja

   Na poslužiteljskoj strani ne smije biti pohranjenog sadržaja, odnosno, svaki zahtjev od strane klijenta prema poslužitelju mora sadržavati dovoljno informacija da bi poslužitelj razumio zahtjev.

3) Predmemorija

   Ovo ograničenje govori kako odgovor na zahtjev mora biti definiran s predmemorijom ili bez nje. Ukoliko je odgovor s predmemorijom, tada klijent ima pravo pohraniti taj odgovor u svoju predmemoriju i iskoristiti ga kasnije za slične zahtjeve.

4) Jedinstveno sučelje

   Pružanjem jedinstvenog sučelja, arhitektura cjelokupnog sustava biva pojednostavljena.

5) Slojeviti sustav

   Arhitektura je organizirana na hijerarhijski način, prema slojevima, tako da niti jedna sastavnica ne može vidjeti dalje od sljedećeg sloja s kojim komunicira.

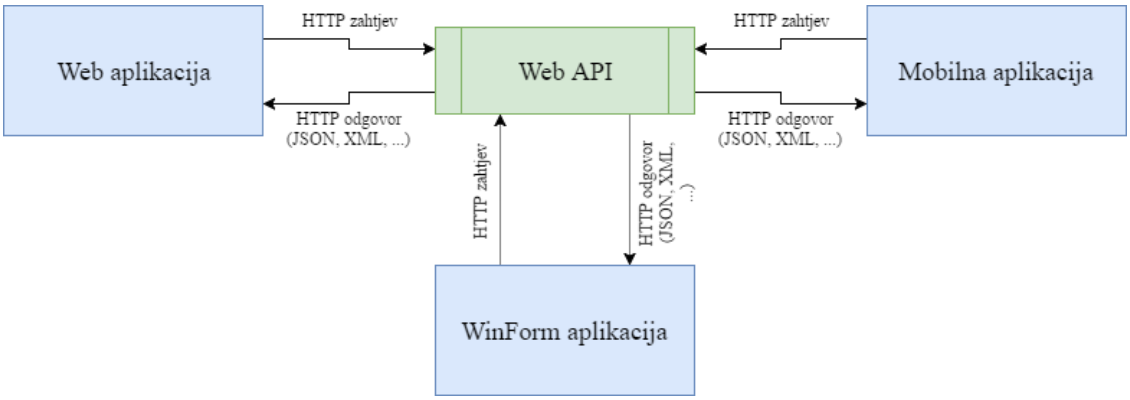
6) Kod na zahtjev

   Ovo ograničenje je neobavezno i omogućuje klijentu da pohrani i izvrši programski kod u obliku skripte.

REST svaku informaciju promatra kao resurs. Resurs može biti slika, dokument, stvarni objekt ili skupina resursa. Svaki resurs ima vlastito stanje, a klijenti se mogu kretati od jednog stanja resursa do drugoga putem hipermedijskih poveznica, koje su sastavni dio reprezentacije resursa [24]. Reprezentacija resursa se, također, sastoji od podataka i metapodataka. Metapodaci su, u suštini, podaci koji daju informaciju o drugim podacima. Medijski tip resursa predstavlja podatkovni format reprezentacije, a njegova je uloga da prenese klijentu informaciju, odnosno, opis reprezentacije, tako da klijent ne mora brinuti o tipu resursa. Prijelaz između stanja odvija se upotrebom glagola za zahtjeve. Najčešće se koriste glagoli HTTP protokola, iako je moguće koristiti postojeće HTTP glagole na drugačiji način, sve dok se poštuje jedinstvenost sučelja. U suprotnome, više se ne može govoriti u REST-u. Glavni glagoli koji se koriste kako bi se opisalo što se želi učiniti s resursima su GET, PUT, POST i DELETE. GET glagol od krajnje točke dohvaća podatke, PUT unosi preinake u postojeće podatke, POST donosi nove podatke na krajnju točku, dok DELETE briše određeni podatak s krajnje točke. Svaki zahtjev se sastoji od zaglavlja i tijela. U zaglavlju se definiraju dodatne informacije u vezi zahtjeva, poput tipa odgovora koji se očekuje. Tijelo zahtjeva služi za prijenos podataka unutar zahtjeva. Primjerice, u slučaju POST glagola u tijelo zahtjeva se prosljeđuje podatak koji se želi pohraniti na krajnju točku u JSON ili XML formatu [25]. Svaki odgovor se sastoji od pripadajućeg tijela i statusnog koda. Tijelo odgovora sadržavat će podatke koji se šalju nazad klijentu na zahtjev, dok statusni kod klijentu daje povratnu informaciju o trenutačnom statusu zahtjeva. Koristeći REST principe, korisnik može stvoriti web usluge, kojima će klijent moći pristupiti i nadopuniti podatke na poslužiteljskoj strani.

3.4 Web API

Prema [26], API (Application Programming Interface) je okruženje ili skup alata koji se koriste za izradu aplikacija i programskih rješenja. Korištenjem API alata, moguće je stvoriti REST usluge, koje predstavljaju sučelje prema podacima određene aplikacije, operacijskog sustava ili drugih usluga. API olakšava izradu modernih višeplatformskih aplikacija na način da razrješava razvojnog programera opširne implementacije programske logike za svaku klijentsku aplikaciju koje komuniciraju s istom bazom podataka. U slučaju programske logike implementirane na pojedinoj klijentskoj aplikaciji, održavanje može postati nepotrebno složeno. Alternativno rješenje je arhitektura s jednim središnjim API-jem, kojeg mogu koristiti sve klijentske aplikacije u svrhu dohvaćanja, nadogradnje i manipulacije podatcima. Prema tome, Web API je API koji je dostupan preko weba, putem HTTP protokola. Na slici 3.4, načinjenoj prema [26], prikazana je upotreba Web API-ja.

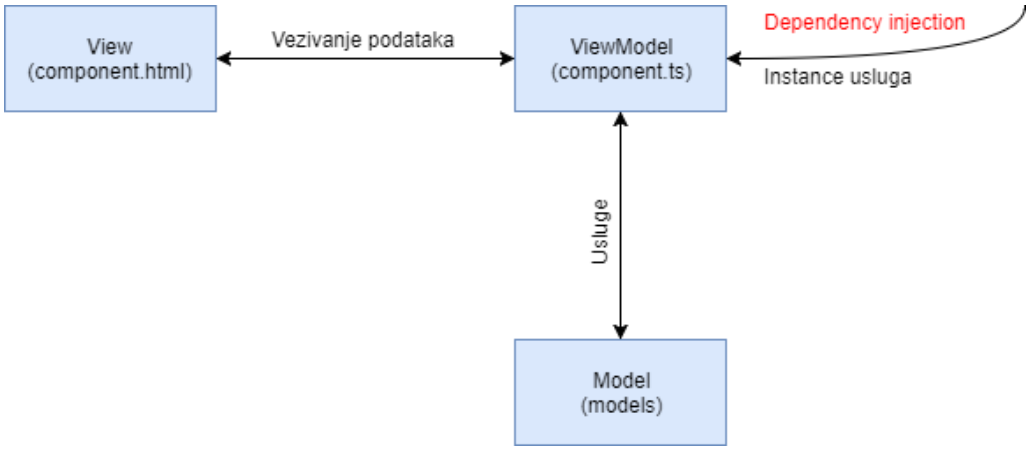


Slika 3.4 Web API

Web API pruža sučelje s kojim mogu komunicirati razne aplikacije, napisane u različitim programskim jezicima. Nadalje, to je idealna platforma za definiranje REST usluga. Komunikacija između klijentske aplikacije i Web API-ja kao krajnje točke odvija se putem HTTP protokola, gdje klijentska aplikacija šalje HTTP zahtjev i očekuje HTTP odgovor kao potvrdu obrađenog zahtjeva za izmjenom ili nadogradnjom podataka, odnosno, resursa ili kao iskoristivi podatak na zahtjev za određenim resursom. HTTP odgovori se mogu formatirati putem Web API *MediaTypeFormattera* u JSON, XML ili bilo koji drugi format koji odgovara korisniku.

3.5 Uzorak MVVM unutar Insa Weba

Uzevši u obzir spomenute tehnologije, kao i strukturu aplikacije InsaWeb prikazane na slici 3.1, moguće je opisati način na koji je MVVM uzorak primijenjen u izradi cjelokupnog rješenja i koju ulogu u tome igraju spomenute tehnologije i pojmovi. Slika 3.5 pruža takav opis.



Slika 3.5 Uzorak MVVM primijenjen na aplikaciju InsaWeb

Kao što je prethodno navedeno u ovome poglavlju, uzorak MVVM nalaže uređenje programskog rješenja u tri dijela. View je dio zadužen za prikaz podataka na korisničkom sučelju i razmještaj elemenata. Također, njegova zadaća je pratiti akcije korisnika, poput klika na gumb ili unosa teksta u polje za unos teksta, te mehanizmom vezivanja podataka obavještavati View Model. Kao što je rečeno u potpoglavlju gdje je opisan okvir Angular, predložak i komponenta međusobno razmjenjuju podatke, događaje i svojstva putem vezivanja podataka. Datoteke sa sufiksom component.html predstavljaju predložak unutar okvira Angular. Model se brine o pohrani podataka i u potpunosti je odvojen od *frontenda* aplikacije. Njegova neovisnost je bitna i zbog toga što računalna i web inačica Inse dijele isti model. Sučelje prema *backendu* u aplikaciji InsaWeb definirano je nizom datoteka u direktoriju pod nazivom *models*. View Model brine se za izgled, međuovisnost i svojstva podatka. Služi kao međusloj između Modela i Viewa, na način da prima informacije o korisničkim akcijama od Viewa te nadograđuje podatke pohranjene na Modelu te na način da organizira podatke iz Modela i prosljeđuje ih tako organizirane prema Viewu, ovisno o potrebi korisnika koji je pokrenuo određenu kontrolu ili zahtjev za podacima. Rječnikom Angulara, View Model je komponenta. View Model unutar aplikacije InsaWeb sačinjavaju datoteke sa sufiksom component.ts, a nalaze se zajedno s datotekama Viewa unutar

direktorija *Components* (prema slici 3.1). View Model komunicira s Modelom preko usluga. Prilikom stvaranja instanci komponenta, odnosno, klasa koje se unutar Angulara nazivaju komponente, pomoću *dependency injectiona* svakoj komponenti je pridružena instanca usluge za dohvaćanje podataka. Usluga za dohvaćanje podataka je REST usluga, kreirana putem Web API-ja.

Uzevši u obzir opisani način rada aplikacije, jasno je kako, uslijed njezine složenosti, postoji veliki potencijal za postojanje previda i grešaka. Razina rizika raste i činjenicom da aplikacija, u svrhu dostupnosti i prilagodljivosti širokom pojasu korisnika i uređaja koje svakodnevno koriste, mora posjedovati jedinstvenu funkcionalnost i prilagodljiv korisnički dizajn za sve platforme preko kojih će joj se u budućnosti pristupati.

4 PRIMJENA TESTIRANJA

4.1 Selenium WebDriver u programskom jeziku C#

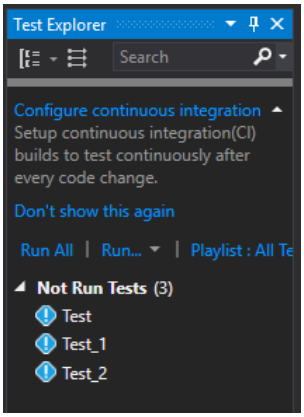
Testiranje aplikacije InsaWeb izvršeno je u Visual Studiu, uporabom programskog jezika C#. Kako bi iskoristivost mogućnosti koje nudi WebDriver bila moguća, potrebno je uključiti Selenium u projekt u Visual Studiu unutar kojega će se definirati testovi. To je moguće napraviti pomoću alata unutar Visual Studia pod imenom Nuget Package Manager. Instalacijom Seleniuma putem Nuget Package Managera, u popis referenci projekta se dodaju odgovarajuće DLL datoteke. U svrhu lakšeg praćenja testova, korišten je okvir za testiranje pod nazivom NUnit. NUnit je okvir za provedbu unit testova. Iako se u ovome slučaju ne provode unit testovi, njegova svojstva olakšavaju organiziranje i praćenje testova za ispitivanje korisničkog sučelja. Instalacija NUnit okvira za testiranje se također izvodi preko Nuget Package Managera, čime se u popis referenci dodaju potrebne DLL datoteke. Kako bi NUnit određeni dio koda prepoznao kao test, potrebno je prije samog dijela koda postaviti oznaku prikazanu u programskom kodu 4.1.

```
[Test]
public void Test_2() {
...
}
```

Programski kod 4.1 Označavanje početka testa

Kada se dio koda označi kao test, tada je on vidljiv u prozoru Test Explorer, kao što je prikazano na slici 4.1.

U popisu testova bit će prikazani svi testovi u trenutačnom projektu s odgovarajućom oznakom, ovisno o tome je li prethodno pokrenut i o tome je li se izvršio do kraja. Također, na dnu prozora je nakon završetka izvođenja testa dostupna informacija o preprekama na koje je test naišao tijekom izvođenja.



Slika 4.1 Test Explorer u Visual Studiu

4.1.1 WebElement

Prema [27], IWebDriver je sučelje koje omogućava korisniku kontroliranje elemenata web stranica preko objektno orijentiranog programskog jezika. Objekti koji se stvaraju kao instance sučelja IWebDriver sadrže metode kojima se određuju osnovne postavke vezane za test. Primjer je dan u kodu 4.2.

```
IWebDriver driver;
driver = new RemoteWebDriver(
 new Uri("http://hub-cloud.browserstack.com/wd/hub/"), capability
);
driver.Navigate().GoToUrl("https://mail.insa-online.com/InsaNgPbz");
```

Programski kod 4.2 Metoda driver objekta za učitavanje web stranice

Metoda `Navigate()` je ugradbena metoda driver elementa, koja se koristi za odlazak na drugi prozor ili za učitavanje nekog URL-a. Ostale metode koje su dostupne unutar sučelja IWebDriver su *Title(), PageSource(), Quit()* i slične. Te naredbe vezane su za web preglednik i prozore web preglednika te nisu izravno vezane za elemente web stranice.

U svrhu kontrole elemenata web stranice Selenium proviđa tip pod imenom WebElement, koji je podskup WebDrivera. On predstavlja HTML element stvarne web stranice. IWebElement je sučelje koje omogućuje izvršavanje određenih operacija nad HTML elementima web stranice. Kako bi to bilo moguće, potrebno je prvo pronaći i dohvatiti određeni HTML element. Način na koji je to moguće odraditi jest putem naredbe koja pripada sučelju IWebDriver, *FindElement()*. Kod 4.3 upravo to prikazuje.

```
var userNameField = driver.FindElement(By.Id("usr"));
```

Programski kod 4.3 Dohvaćanje HTML elementa pomoću WebDrivera

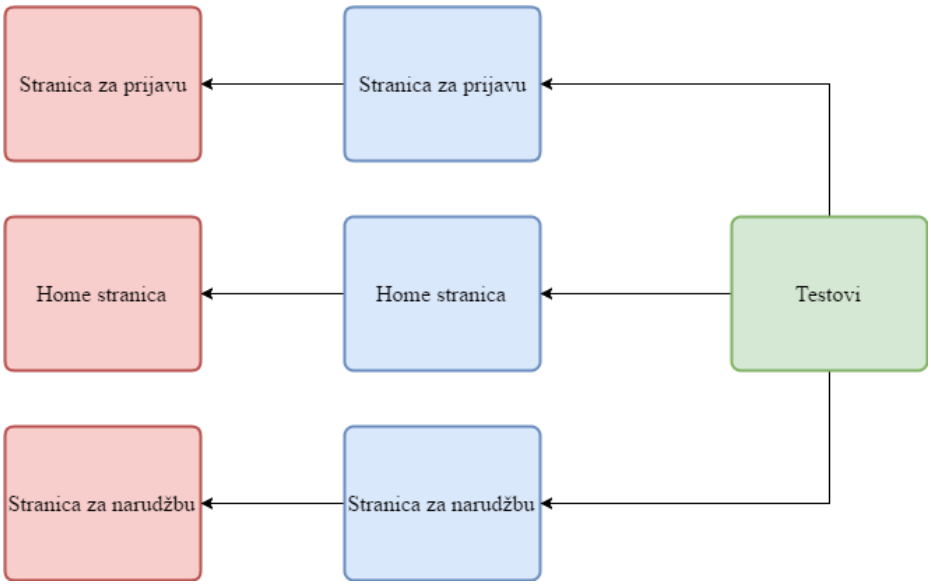
Kao što stoji u prethodnom kodu, određeni element se upotrebom metode *FindElement()* pronalazi i dohvaća po kriteriju koji je određen preko *By* objekta, kojeg metoda prima kao ulazni argument. U ovom slučaju to je pronalaženje putem ID-a elementa. Povratni tip spomenute metode je IWebElement, koji je spremljen u varijablu pod nazivom `userNameField`.

Svaki IWebElement pruža niz ugrađenih metoda koje se mogu izvršiti nad dohvaćenim elementima web stranice. Iako su metode dostupne prilikom pisanja programskog koda, prilikom izvršavanja testa će doći do pogrešaka ako se na neki element pokuša primijeniti metoda koja opisuje akciju koju je nemoguće izvršiti na takvom HTML elementu. Primjerice, metoda *Clear()* dostupna je prilikom pisanja koda za WebElement koji predstavlja gumb na web stranici, iako ona definira akciju brisanja unosa iz polja za unos teksta. Očito je kako su ta metoda i navedeni WebElement nespojivi. Visual Studio neće javiti pogrešku prilikom kompiliranja koda, ali unutar prozora Test Explorer bit će dostupan opis pogreške, a test se neće završiti i bit će označen crvenom bojom. Neke od dostupnih metoda sučelja IWebElement su *SendKeys(),* za unos teksta u tekstualno polje, *Click(),* za klik lijevom tipkom miša na gumb ili poveznicu, *Selected{get;}*, za prepoznavanje označenog HTML elementa, *Size{get;}* za povratnu informaciju o veličini elementa i druge metode. Metode koje pruža sučelje IWebElement su raznolike i pružaju testeru mogućnost potpune simulacije korisničke interakcije s web aplikacijom. Međutim, pisanje testova na način da se slijed naredbi izvršava za svaki pojedini element, od lociranja i dohvaćanja do manipulacije njime, postaje problem ukoliko se sve odvija u istoj datoteci. Kod postaje nepregledan, dugačak i neučinkovit. Kako bi se riješio taj problem i testiranje putem Selenium WebDrivera i njegovih sastavnica što više približio objektno orijentiranom programiranju i njegovim principima, razvijen je PageObject.

4.2 Uzorak PageObject

PageObject je uzorak za dizajniranje automatiziranih testova korištenjem Seleniuma [28]. On pruža pogled na web aplikaciju na način da ju razdvaja na stranične objekte, gdje su stranični objekti zapravo elementi stranice web aplikacije, iako je moguće i određeni funkcionalni dio stranice predstaviti kao stranični objekt. Razvoj takvog uzorka započeo je prepoznavanjem potrebe da programski tester mora vidjeti i učiniti sve što može i čovjek. Uzorak PageObject se konstruira klasama koje su međusobno povezane, a njihova povezanost predstavlja logičku vezu između stvarnih stranica web aplikacije. Na svakoj klasi je zadatak da pronađe web

elemente pripadajuće stranice te da pruži sučelje za manipulaciju web elementima, to jest, skupinu metoda za rukovanje istima, poštujući princip enkapsulacije. Načini dohvaćanja određenog web elementa i izvršavanja operacija nad njima trebaju biti enkapsulirani unutar klase, a prema van se trebaju izložiti samo rezultati spomenutih operacija. Primjerice, ako je potrebno pronaći tekst određenog zaglavlja, tada je zadaća klase PageObjecta pronaći to zaglavlje i izložiti metodu za dohvaćanje čija je povratna vrijednost *string*. Slika 4.2 olakšava razumijevanje navedenog.



Slika 4.2 Model PageObject

Pri definiranju automatiziranih testova, pojedini testovi imaju pristup jedino metodama koje su izložene kroz sučelje svakog PageObjecta. Primjerice, aplikacija za trgovinu putem interneta se sastoji od nekoliko osnovnih dijelova, odnosno, stranica. Svaka korisnički usmjerena web aplikacija sadržavat će stranicu za prijavu, gdje se od korisnika traži da unese svoje podatke kako bi pristupio punom sadržaju aplikacije. Njezini elementi su nekoliko polja za unos teksta te gumb za potvrdu. Također, svaka će web aplikacija sadržavati i *home* stranicu, odnosno, stranicu na kojoj su izložene osnovne informacije i poveznice prema drugim stranicama. Primjera radi, aplikacija za trgovinu putem interneta će imati i stranicu na kojoj se rukuje narudžbama, potvrđuje narudžba ili briše artikl s popisa narudžbe. Poštujući uzorak PageObject, svaka od navedenih stranica će imati pripadajuću klasu identičnog imena, koja će služiti za pristup elementima stranice i izlaganje metoda za rukovanje elementima stranice. Uz osvrt na dani primjer, klasa stranice za prijavu će biti odgovorna za identifikaciju polja za unos teksta i gumba za potvrdu te će biti odgovorna za izlaganje metode koja će kao

argumente primiti korisničke podatke, unijeti ih u polja za unos i pritisnuti gumb za potvrdu. Izložene metode pojedinih klasa mogu koristiti različiti testovi, budući da klase stranica nisu strogo vezane za pojedini test, što je i cilj ovog uzorka. U duhu spomenutog primjera, tester može sastaviti dva testa, od kojih će jedan provjeriti ponašanje stranice za prijavu u slučaju ispravnog unosa korisničkih podataka, dok će drugi provjeriti ponašanje za unos pogrešnih podataka. Oba testa mogu preko vlastite instance klase stranice za prijavu pristupiti izloženoj metodi koja će odraditi unos podataka i pritisak gumba za potvrdu. Neke od prednosti uzorka PageObject su smanjenje umnažanja koda, čitljivost i robusnost testova te visoki stupanj održivosti testova, pogotovo u projektima temeljenim na agilnim metodologijama, gdje se izgled i raspored web elemenata brzo i često mijenja.

4.2.1 PageFactory u C#

Dodatak uzorku PageObject za programski jezik C# poznat je pod nazivom PageFactory. U suštini, to je optimizirani koncept modela PageObject namijenjen za Selenium WebDriver. Koristi se za incijalizaciju elemenata straničnog objekta bez upotrebe naredbi *FindElement* i *FindElements*. Kao alternativa tome, PageFactory nudi pronalaženje elemenata preko anotacije *FindsBy*. Ona kao argument prihvaća ime oznake, ime, ID, ime klase, Css, tekst poveznice ili Xpath, prema [28]. Primjer takvog načina pronalaženja elementa na web stranici je dan u kodu 4.4.

```
[FindsBy(How = How.Name, Using = „ime“)]
```

Programski kod 4.4 Upotreba anotacije *FindBy* putem imena elementa

Ono što je važno napomenuti u vezi koda 4.4 jest to da se oznaka mora nalaziti iznad varijable u koju se želi pospremiti rezultat potrage za stvarnim elementom na web stranici.
Naredba za inicijaliziranje instance PageObject klase glasi *InitElements* i prima dva ulazna parametra. Pozivanje ove naredbe obavlja se u konstruktoru određene klase. Ukoliko nije moguće stvoriti instancu određene klase, bit će podignuta iznimka.

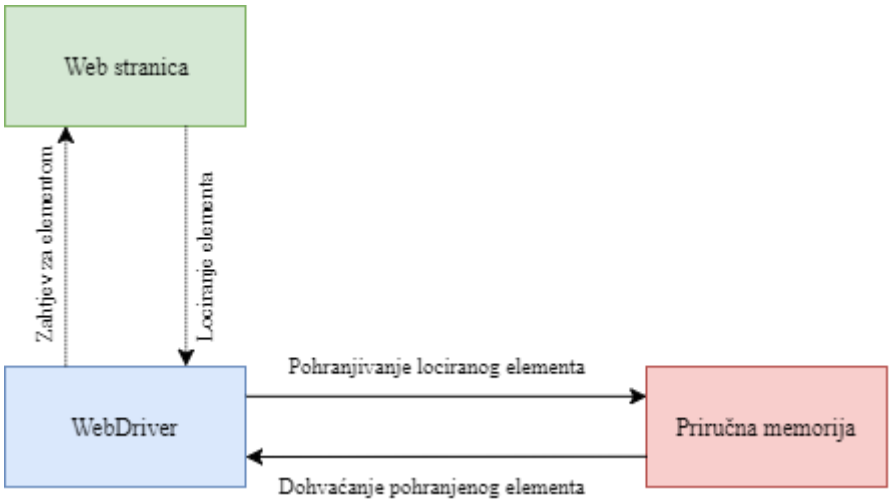
```
PageFactory.InitElements(WebDriver, PageObject)
```

Programski kod 4.5 Izgled naredbe *InitElements*

Prema kodu 4.5, vidljivo je kako ova naredba kao argumente prima instancu WebDriver-a, koji se koristi za identifikaciju elemenata stranice te instancu trenutne klase koju se želi inicijalizirati.

4.2.2 CacheLookup

U slučaju kada su scenariji testova dugački i opsežni te sadrže ponavljanje akcija, što podrazumijeva rukovanje s istim elementima web stranice, WebDriver će potražiti iste elemente stranice svaki put kada se pozove metoda koja obavlja određene operacije nad njima, što može znatno usporiti proces izvršavanja testa. U aplikacijama koje koriste tehnologiju Ajax, gdje se elementi stranice mogu pojaviti nakon neke Ajax akcije, poželjno je da WebDriver svaki put iznova traži elemente koje koristi metoda koja se poziva prilikom izvršavanja testa. Međutim, ako je poznato da se neki elementi stranice neće mijenjati prilikom korisničke interakcije sa njenim sadržajem, tada je nepraktično da se, svaki puta kada se izvršava dio testa u kojemu se koristi ciljani element, iznova traži isti. U tu svrhu postoji oznaka *CacheLookup,* koja uparena s određenim elementom govori naredbi *InitElement* da traženi element, nakon što ga WebDriver nađe, pospremi u priručnu memoriju. To znači da će neki web element biti pronađen samo jednom, pospremljen u pričuvnu memoriju i sljedeći puta kada se izvrši neka operacija nad njime, neće se morati ponovno tragati za njime, nego će biti odmah dostupan, kao što je prikazano na slici 4.3.



Slika 4.3 Princip *CacheLookup*

Kod 4.6 prikazuje jednostavnu primjenu *CacheLookup*-a.

```
[FindsBy(How = How.Name, Using = „ime“)]
[CacheLookup]
```

Programski kod 4.6 Atribut *CacheLookup*

4.2.3 Enkapsulacija

Budući da je pojam enkapsulacije neizbježan u primjeni uzorka PageObject, zajedno s načinom definiranja varijabli i metoda unutar klase pripadajuće web stranice, bitno ga je pojasniti te dati pregled načina na koji se ostvaruje. Princip enkapsulacije je jedan od ključnih principa objektno orijentiranog programiranja, koji omogućuje veću čitljivost koda, njegovu sigurnost i održivost. Enkapsulacija je mehanizam kojim se podaci i programska logika koja radi s istim podacima vežu zajedno u istu jedinicu, kao što je navedeno u [29]. Preciznijom definicijom može se govoriti o varijablama umjesto podataka i o metodama umjesto programske logike. Uzimajući prethodnu definiciju u obzir, varijable određene klase ne smiju biti dostupne izvan te iste klase, odnosno, one moraju biti sakrivene od ostalih klasa. Pristupiti im se može jedino preko metoda koje su izložene, također u istoj klasi. Kako bi to bilo ostvarivo, potrebno je varijable deklarirati kao privatne, modifikatorom `private` te prirediti metode pristupa i promjene u svrhu čitanja podataka i postavljanja njihovih vrijednosti.

Metoda pristupa je metoda kojom se postavlja upit objektu o njegovom stanju. Česta je uporaba metoda pristupa nad svojstvima. Svojstva imaju svoju metodu pristupa pod nazivom *get*, međutim, metoda pristupa ne mora biti ograničena samo na svojstva.

Metoda promjene je javna metoda koja se koristi za promjenu stanja objekta. Ona skriva način na koji se odvija promjena stanja. Kao i metoda pristupa, karakteristična je za svojstva pod nazivom *set*.

Prema navedenim smjernicama, svaki stranični objekt će se deklarirati modifikatorom `private` te prikazati kao svojstvo odgovarajućom sintaksom, putem metoda pristupa i promjene, kao što je prikazano na kodu 4.7.

```
[FindsBy(How = How.Name, Using = „ime“)]
[CacheLookup]
private IWebElement NameField { get; set; }
```

Programski kod 4.7 Svojstvo *NameField* s metodama pristupa i promjene

Nakon prilagodbe varijabli mehanizmom enkapsulacije, potrebno je definirati metode kojima se manipulira svojstvima. Budući da su sva svojstva unutar enkapsulirane klase privatne, potrebno je takve metode definirati također unutar klase. Primjer slijedi u programskom kodu 4.8.

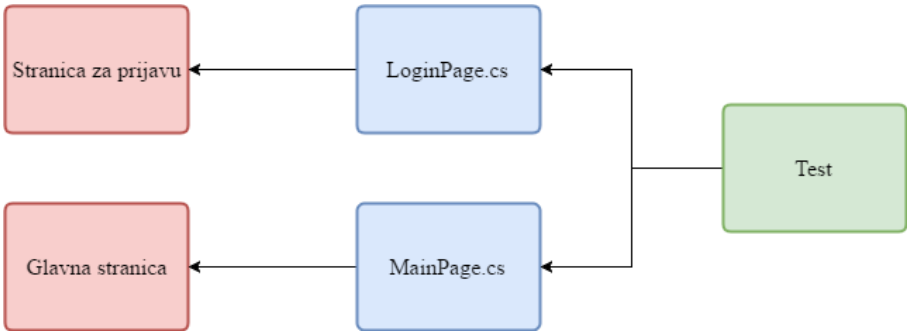
```
public void UnesiIme(string ime)
{
NameField.SendKeys(ime);
}
```

Programski kod 4.8 Metoda za unos teksta u polje *NameField*

Iz programskog koda 4.8 vidljivo je kako se za pristup svojstvu `NameField`, čija je deklaracija prethodno prikazana u kodu 4.7, definira posebna metoda pod nazivom `UnesiIme`, koja je javnog tipa, odnosno, dostupna iz drugih klasa. Vrlo bitan princip pri definiranju metoda za manipulaciju straničnih objekata jest definiranje samo onih metoda koje će se stvarno koristiti u testu. Također, bitno je definirati jednu metodu koja će odraditi određeni posao s elementima stranice, a koja će sadržavati uzastopne akcije nad elementima. Na taj način se izbjegava pojava velikog broja metoda u enkapsuliranoj klasi koje čine mali posao, što pruža rješenje za jedan od glavnih problema koji je potaknuo razvoj enkapsulacije, a to je smanjenje obujma koda.

4.3 Implementacija modela PageObject u testnom projektu

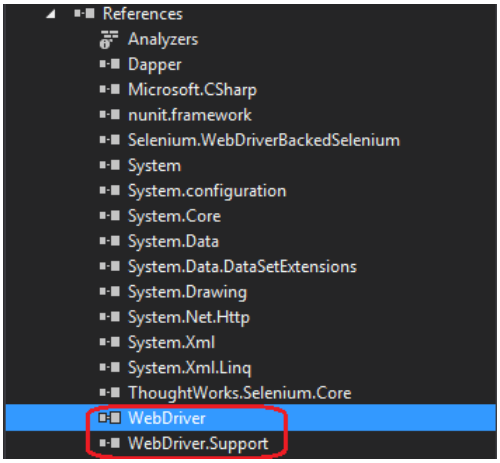
Aplikacija InsaWeb je jednostranična aplikacija. Međutim, korisnik se prvo susreće sa stranicom za prijavu, koja služi kao ulazna točka za aplikaciju. Razlog tomu je što samo unaprijed registrirani korisnici kojima je dodijeljeno korisničko ime i lozinka mogu pristupiti aplikaciji. Iako stranica za prijavu tehnički nije dio aplikacije, svaki korisnik će prilikom pokušaja dohvaćanja glavne stranice naići na stranicu za prijavu. Zbog karaktera testiranja po scenariju, koje zagovara stavljanje testera u položaj korisnika i oponašanje predvidivih akcija korisnika, ključno je u svaki scenarij testiranja uključiti i rukovanje s elementima stranice za prijavu. Prema tome, za potrebe testiranja InsaWeb se promatra kao dvostranična aplikacija. Sukladno tome, stvorene su dvije klase prema uzorku PageObject, koje predstavljaju stranicu za prijavu i glavnu stranicu aplikacije. Prema slici 4.2, načinjena je slika 4.4, koja točnije dočarava strukturu uzorka PageObject za tekuću konkretnu primjenu.



Slika 4.4 Testni projekt prema modelu PageObject

Kao što je rečeno, postoje dvije stranice unutar testne web aplikacije. Prema uzorku PageObject, načinjene su dvije klase od kojih je svaka zadužena za pronalazak elemenata web stranice i tretiranje istih kao stranične objekte. Budući da se na stranici za prijavu nalaze samo dva polja za unos i jedan gumb, manji posao odrađuje klasa LoginPage.cs. Klasa MainPage.cs je puno složenija po broju elemenata koji se dohvaćaju i po broju metoda koje su definirane za rukovanje s elementima. Tester može stvoriti velik broj testova, odnosno, klasa, koje imaju pristup javnim metodama klasa odgovarajućih stranica. Ovisno o scenarijima testova poželjno je stvoriti više različitih klasa u kojima će biti definirani pojedini testovi.

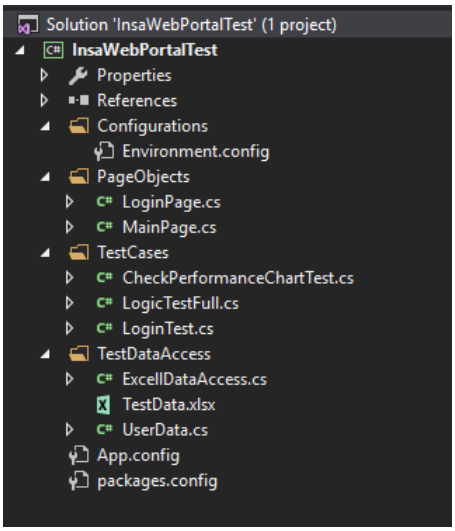
Kako bi navedene klase mogle dohvaćati web elemente sa stvarnih web stranica, potrebno je koristiti Selenium WebDriver, koji se može dodati u projekt kao proširenje Visual Studia. Potrebno je da DLL datoteka WebDrivera bude uvrštena u popis referenci projekta, kao na slici 4.5.



Slika 4.5 WebDriver DLL

Preko WebDrivera klasa komunicira s web preglednikom i traži željene elemente na web stranici, promatrajući ih kao objekte. Radi potpunog i jasnog razumijevanja načina

organiziranja projekta testiranja aplikacije InsaWeb, slijedi prikaz projektnog stabla u Visual Studiu. Svi daljnji opisi će se pozivati na sliku 4.6.



Slika 4.6 Projektno stablo testnog projekta

Mapa pod nazivom PageObjects sadržava već spomenute klase LoginPage.cs i MainPage.cs. U njima su definirani stranični objekti te metode koje služe za manipulaciju njima. Primjer dohvaćanja elementa sa stranice za prijavu prikazan je u kodu 4.9.

```
[FindsBy(How = How.Name, Using = “password”)]
[CacheLookup]
private IwebElement Password { get; set; }
```

Programski kod 4.9 Dohvaćanje polja za unos lozinke

U upravo navedenom dijelu koda, prikazan je način dohvaćanja polja za unos lozinke. Polje je dohvaćeno putem naziva elementa te je rezultat potrage za elementom pohranjen u priručnu memoriju. Element web stranice je unutar klase pohranjen u obliku privatnog svojstva s pripadajućim metodama pristupa i promjene pod nazivom `Password`. Ostali elementi se dohvaćaju na isti način, bez obzira radi li se o polju za unos teksta, gumbu ili nekom drugom elementu. Kod 4.10 to jasno pokazuje.

```
[FindsBy(How = How.Id, Using = “login_btn”)]
[CacheLookup]
private IwebElement Submit { get; set; }
```

Programski kod 4.10 Dohvaćanje gumba za prijavu

Vidljivo je kako se dohvaćanje različitih elemenata web stranice ne razlikuje po sintaksi. Jedina razlika u dohvaćanju polja za unos lozinke i gumba za prijavu je način dohvaćanja elementa sa web stranice. U kodu 4.10 korišteno je dohvaćanje putem ID-a elementa. Način

dohvaćanja se mijenja od elementa do elementa, ovisno o tome koje je podatke o elementu ostavio dizajner prilikom dizajniranja korisničkog sučelja.
U programskom kodu 4.11 prikazan je način definiranja metode koja manipulira dohvaćenim elementima stranice za prijavu.

```
Public void LoginToApplication(string testName)
{
  var userData = ExcellDataAccess.GetTestData(testName);
  UserName.SendKeys(userData.Username);
  Password.SendKeys(userData.Password);
  Submit.Submit();
}
```

Programski kod 4.11 Metoda za prijavu na InsaWeb

Metoda prikazana u programskom kodu 4.11 obavlja prijavu za ulaz na glavnu stranicu InsaWeba, raspolažući pri tome korisničkim podacima, to jest korisničkim imenom i lozinkom. Za dohvaćanje korisničkih podataka koristi se posebna tehnika koja će biti opisana kasnije u radu. Polje za unos korisničkog imena te polje za unos lozinke sadrže ugrađene funkcije za predviđene akcije koje se mogu izvršiti nad elementima web stranice. Jedna od njih je i *SendKeys()*, koja služi upravo za upisivanje teksta u polje za unos teksta. Pozivom ugrađene funkcije *Submit()* na gumbu, WebDriver će obaviti akciju ekvivalentnu pritiskanju gumba mišem.
Klasa MainPage.cs sadrži veću raznolikost od LoginPage.cs iz razloga što se na njoj događa sva dinamika uz mnoštvo različitih web elemenata. Jedna od mogućnosti pri dohvaćanju elemenata stranice jest pronalaženje višestrukih elemenata i pospremanje u listu web elemenata, kao što je prikazano u kodu 4.12.

```
[FindsBy(How = How.CssSelector, Using = „.k-button.k-button-icontext.k-grid-
security“)]
[CacheLookup]
private Ilist<IWebElement> EquitiesSecurityButtons { get; set; }
```

Programski kod 4.12 Dohvaćanje višetrukih gumbova

U ovom primjeru, dohvaćanje se vrši putem CssSelectora, odnosno, cilj dohvaćanja je nad skup višestrukih gumbova. Na taj način može se pristupiti višestrukim elementima odjednom. Identificirani elementi spremaju se u listu i u kontekstu metoda im se pristupa jednako kao i elementima bilo kakve druge liste u programskom jeziku C#. Primjena takvih lista je korisna u testnim scenarijima kada se na korisničkom sučelju nalaze višestruki gumbovi koji se nalaze blizu jedan drugome te se klikom na njih obavlja slična funkcionalnost. Većinom se radi o situaciji kada tester želi ispitati funkcionalnost svih gumbova tako da klikne na svakog

od njih. Tada se takvim elementima pristupa u petlji koja prema indeksu liste izlaže određenoj akciji pojedini gumb.

Poseban način definiranja metode za rukovanje elementom web stranice događa se pri susretu s kontrolom povlačenja i ispuštanja. Rukovanje povlačenjem i ispuštanjem elementa odvija se putem klase tipa `Actions`, koja ima ugrađene funkcije `ClickAndHold`, kojoj se predaje naziv elementa kojega se želi povući, zatim `MoveToElement`, koja kao argument prima objekt na čije koordinate se želi dovući element koji se pomiče i `Release`, kojom se ispušta povlačeni element na željeno odredište. Primjena toga vidljiva je na programskom kodu 4.13.

```
internal void DragFilterHeaderPortfolio()
{
  Actions builder = new Actions(driver);

  Iaction dragAndDrop = builder.ClickAndHold(PortfolioHeaders[0])
       .MoveToElement(PortfolioDropdownHeader)
       .Release(PortfolioDropdownHeader)
       .Build();

  dragAndDrop.Perform();
 }
```

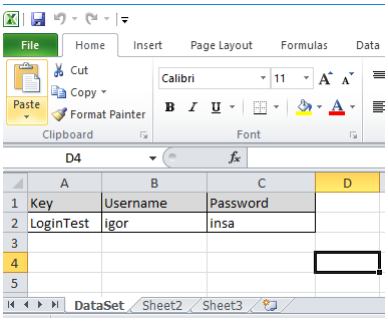
Programski kod 4.13 Metoda za povlačenje i ispuštanje elemenata zaglavlja tablice

Metoda `DragFilterHeaderPortfolio` se koristi za simuliranje korisničkog rukovanja elementima koji podržavaju povlačenje i ispuštanje klikom i pokretima miša. U konkretnom slučaju, ti elementi su elementi zaglavlja tablice za prikaz podataka o portfeljima, koji služe kao filteri podataka te se slaganjem njihovog rasporeda u zaglavlju odabire prioritet filtera.

Pri testiranju prijave višestrukih korisnika nije poželjno držati korisničke podatke u programskom kodu. Bolji je princip učitavanja tih podataka iz vanjske datoteke. Vanjske datoteke iz kojih se učitavaju podaci mogu biti raznih formata od kojih su neki Excel datoteke, ADO objekti, CSV datoteke i drugi. Za potrebe testiranja aplikacije InsaWeb korištena je Excel datoteka. Prednost testiranja korištenjem vanjskih izvora podataka je isti argument kao i za primjenu enkapsulacije, a to je smanjenje obujma programskog koda. U slučaju da su podaci spremljeni u programskom kodu, promjenom podataka se mora promijeniti i kod. Korištenjem Excel datoteke kao vanjskog izvora korisničkih podataka, ukoliko dolazi do potrebe izmjene, dodavanja ili brisanja podataka, nema potrebe za mijenjanjem programskog koda. Sve izmjene se provode unutar tabličnog kalkulatora Microsoft Excel.

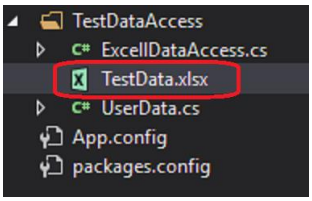
U svrhu upravo opisanog principa, stvorena je datoteka pod nazivom TestData.xlsx, u koju će biti pospremljeni svi korisnički podaci koji će se koristiti za svrhe testiranja funkcionalnosti

prijave na glavnu stranicu InsaWeba. Naziv kartice je promijenjen u DataSet te je načinjena tablica s imenima kolona Key, Username i Password. Izgled opisanog uređenja vidljiv je na slici 4.7.



Slika 4.7 Korisnički podaci u tablici Excel

Nakon uređenja tablice i unosa podataka, uređena Excel datoteka dodana je u projektno stablo unutar mape pod nazivom TestDataAccess, kao što je vidljivo na slici 4.8.



Slika 4.8 Datoteka TestData.xlsx unutar projektnog stabla

Također, sa slike 4.8 je vidljivo kako unutar mape TestDataAccess postoje i druge datoteke u kojima je definiran način dohvaćanja podataka iz Excel datoteke. Klasa pod nazivom UserData sadržava javna svojstva tipa *string*, koja nazivom odgovaraju kolonama u tablici koja sadržava korisničke podatke. Kod 4.14 upravo to prikazuje.

```
Class UserData
    {
        public string Key { get; set; }
        public string Username { get; set; }
        public string Password { get; set; }

    }
```

Programski kod 4.14 Klasa UserData

U datoteci ExcelDataAccess.cs definiran je pristup Excel datoteci i izvlačenje podataka iz nje putem konekcije OleDB. OleDB je poseban API namijenjen dohvaćanju podataka iz zasebnih datoteka. Za dohvaćanje podataka putem OleDB, potrebno je u programskom kodu otvoriti konekciju OleDB, izvršiti određeni upit, pospremiti rezultat upita te zatvoriti konekciju. U

kodu 4.15 prikazane su dvije linije koda zadužene za upit nad podacima te spremanje rezultata.

```
Var query = string.Format(„select * from [DataSet$] where key='{0}'“, keyName);
var value = connection.Query<UserData>(query).FirstOrDefault();
```

Programski kod 4.15 Upit nad podacima i pohrana rezultata

U prvoj liniji programskog koda odrađen je upit nad podacima koji se nalaze unutar kartice pod nazivom DataSet, a to je upravo naziv koji je dodijeljen kartici unutar TestData.xlsx datoteke. Pretraživanje podatka vrši se prema vanjskom podatku pod imenom `keyName`, a on odgovara vrijednosti upisanoj u prvoj koloni izrađene tablice. U drugoj liniji se rezultat upita sprema u varijablu `value`, koja će vratiti objekt tipa UserData.

Prema slici 4.6 koja prikazuje stablo testnog projekta, vidljivo je kako se unutar mape TestCases nalaze klase u kojima su definirani različiti testni scenariji. U svrhu pojašnjenja korištenja metoda koje rukuju dohvaćenim elementima web stranice, u nastavku slijede primjeri programskog koda u kojima se to odvija. Prvi korak u definiranju svakog testa je kreiranje instance sučelja tipa IWebDriver, kao što je prikazano u programskom kodu 4.16.

```
IWebDriver driver;
driver = new RemoteWebDriver(
  new Uri(„http://hub-cloud.browserstack.com/wd/hub/“), capability
);
```

Programski kod 4.16 Stvaranje instance sučelja IWebDriver

Kao što je spomenuto ranije, WebDriver je sučelje prema web stranici preko kojega je omogućeno dohvaćanje pojedinačnih elemenata stranice. Stoga se prilikom stvaranja nove instance klase koja predstavlja web stranicu, njoj prosljeđuje upravo stvorena instanca WebDrivera. To je prikazano u programskom kodu 4.17.

```
var loginPage = new LoginPage(driver);
```

Programski kod 4.17 Stvaranje instance stranične klase

Klasa LoginPage.cs u svom konstruktoru svojoj lokalnoj varijabli tipa IWebDriver pridružuje proslijeđenu instancu te je koristi pri inicijalizaciji stranice. U programskom kodu 4.18 vidljiv je način na koji se poziva metoda za unos korisničkih podataka.

```
loginPage.LoginToApplication(„LoginTest“);
```

Programski kod 4.18 Poziv metode za unos korisničkih podataka na stranici za prijavu

Pozivu metode za unos korisničkih podataka prosljeđuje se ključ pod kojim su u datoteci TestData.xlsx uneseni korisničko ime i lozinka koje se želi upotrijebiti pri pokušaju prijave na stranicu.

Na jednak se način stvara instanca klase glavne stranice te pozivaju njezine metode. Jasno je kako se korištenjem uzorka PageObject za definiranje testova postiže jasnoća i preglednost koda te se lako prate koraci zadanog testnog scenarija, budući da nakon postavljanja početnih postavki u klasi testnog scenarija slijede pozivi metoda čija su imena opisnog karaktera. Drugim riječima, prateći linije koda i metode koje se pozivaju, lako je moguće iščitati zadani scenarij testa. Pri završetku testnog scenarija potrebno je pozvati metodu iz programskog koda 4.19, kojom se zatvaraju svi prozori web preglednika koje je otvorio WebDriver.

```
Driver.Quit();
```

Programski kod 4.19 Poziv metode za zatvaranje prozora preglednika

Sa završenim opisom programskog rješenja automatiziranih testova, slijedi opis načina pokretanja testova na Browserstacku te pregled dobivenih rezultata i načina rješavanja izazova na tom području.

5 REZULTATI TESTOVA S ANALIZOM

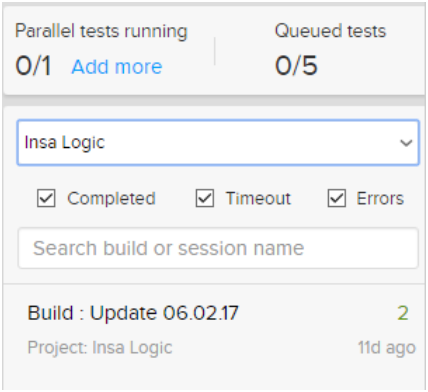
5.1 Pregled mogućnosti Browserstacka

Testovi napisani na način opisan u prethodnom poglavlju, odnosno, koristeći uzorak PageObject, provedeni su na usluzi oblaka računala pod nazivom Browserstack, koja je opisana u prvom poglavlju ovog rada. Browserstack omogućuje pokretanje testova na više različitih preglednika. Način na koji se test napisan u Seleniumu pokreće na platformi Browserstack je prikazan na slici 5.1 i prikladno objašnjen.

```
DesiredCapabilities capability = new DesiredCapabilities();
capability.SetCapability("browserstack.user", "bernardinkatic1");
capability.SetCapability("browserstack.key", "p2KbRBW5s7RASRxXTrHX");
capability.SetCapability("project", "Insa Logic");
capability.SetCapability("build", "Update 06.02.17");
capability.SetCapability("browser", "IE");
capability.SetCapability("browser_version", "11.0");
capability.SetCapability("os", "Windows");
capability.SetCapability("os_version", "7");
capability.SetCapability("resolution", "1920x1080");
capability.SetCapability("browserstack.debug", "true");
capability.SetCapability("browserstack.local", "false");
```

Slika 5.1 Odabir mogućnosti Browserstacka

Način na koji se test napisan pomoću Seleniuma pokreće na Browserstacku je sadržan u nekoliko redaka koda. Prva stvar koju je potrebno napraviti je stvoriti objekt tipa *DesiredCapabilities*, koji se nalazi unutar nazivlja OpenQA.Selenium.Remote. Novostvoreni objekt sadrži metodu pod nazivom *SetCapability*(), preko koje tester komunicira s Browserstackom unutar razvojnog okruženja u kojem je pisao test. U ovom slučaju to je okruženje .NET. Budući da je Browserstack alat čije se usluge naplaćuju, svaki korisnik dobiva svoje korisničko ime i lozinku preko koje može pristupiti internetskim stranicama usluge, ali i preko kojih jedino može izložiti svoj test. Nakon unosa korisničkog imena i lozinke, slijede redci kojima se organiziraju testovi. Svaki test mora pripadati određenom *buildu*, a *buildovi* mogu biti složeni u projekte. Na testeru je odluka kako će izvršiti podjelu i organizirati svoje testove. Način na koji je ta podjela vidljiva testeru kada pristupi stranicama Browserstacka je prikazan na slici 5.2.



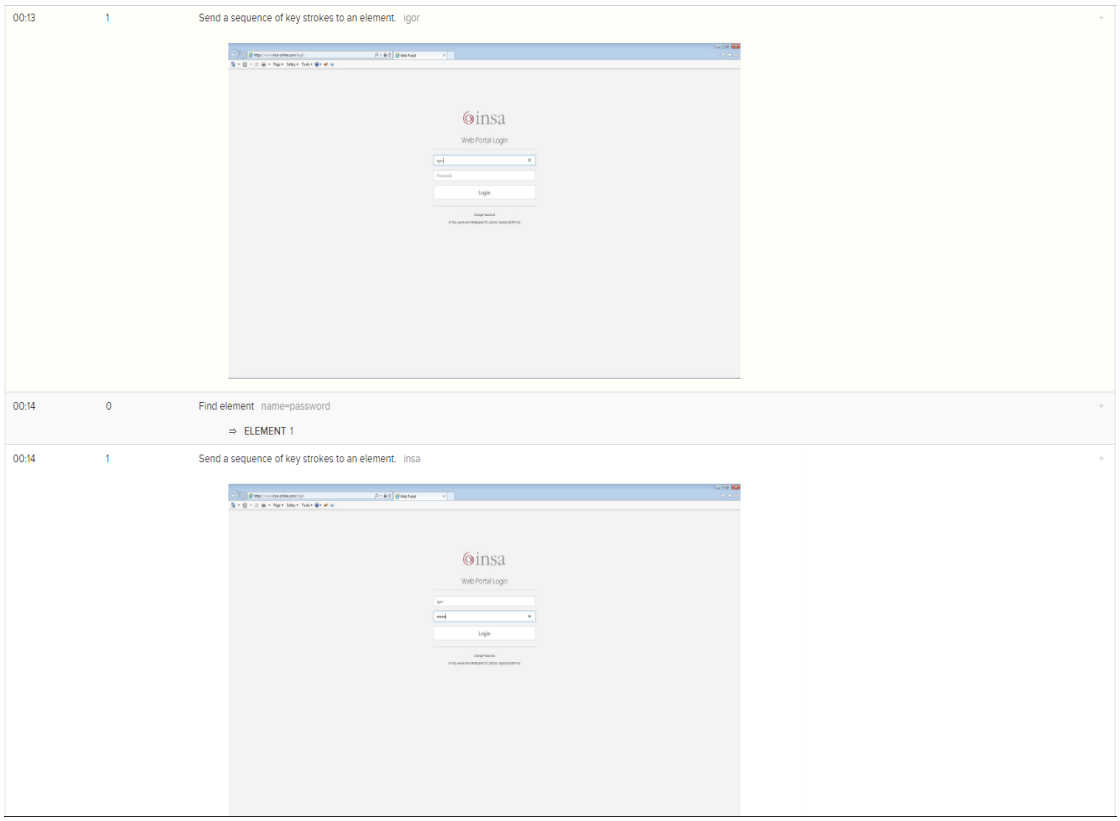
Slika 5.2 Projekti i *buildovi* u Browserstacku

U padajućem izborniku korisnik može odabrati jedan od projekata koje je prethodno započeo, a unutar projekta se nalazi popis stvorenih *buildova* s informacijom o tome koliko je ukupno pokrenutih testova unutar njih te koliko je vremena prošlo od pokretanja zadnjeg testa.

Sljedeći odsječak koda na slici 5.1 sastoji se od pet redaka u kojima se prilagođen preglednik zahtjevima ili potrebama testera. Prvim retkom se odabire preglednik, u ovom slučaju Internet Explorer. Zatim je odabrana inačica istog preglednika. Pri tome je bitno utvrditi koja je inačica preglednika trenutno dostupna jer u slučaju odabire krive inačice, Browserstack će prijaviti grešku i neće samostalno pokrenuti test na nekoj od dostupnih inačica. Slično odabiru preglednika i njegove inačice, potrebno je odabrati operacijski sustav i njegovu inačicu. Dostupni operacijski sustavi su Windows te OS X. Windows nudi sve inačice koje su izašle od XP do Windows10, ne uključujući Vista, dok OS X nudi sve inačice od Snow Leopard do Sierra. Vrlo bitna postavka koju je moguće odabrati u mogućnostima Browserstacka je rezolucija ekrana. Budući da je aplikacija InsaWeb aplikacija čija je odlika responzivni dizajn, kontrolom dimenzija otvorenog prozora preglednika moguće je provjeriti kako će se ponašati elementi korisničkog sučelja pri različitim vrijednostima rezolucije. Predzadnjim retkom koda sa slike 5.1 uključuje se pomoćni alat za otkrivanje pogrešaka, koji je prema izvornim postavkama isključen. Zadnjim retkom se onemogućuje lokalno testiranje, što je potrebno učiniti za potrebe testiranja InsaWeba jer aplikacija nije pokrenuta lokalno, već joj se pristupa preko udaljene adrese.

Posebno koristan je alat za otkrivanje pogrešaka, jer testeru pruža više načina da dobije povratnu informaciju o uspješnosti ili neuspješnosti testa i razlozima potencijalnog neuspješnog ishoda. Kako bi se otkrile pogreške nastale pri pisanju koda za testove, Browserstack nudi sirove zapise, odnosno, zapise konzole, zatim vizualne zapise te tekstualne zapise. Vizualni zapisi sadržavaju snimke ekrana tijekom izvršavanja cjelokupnog testnog

scenarija. Ukoliko je pogreška otkrivena, vizualni zapis prestaje sa snimanjem. Primjer vizualnog zapisa testa u Browserstacku je vidljiv na slici 5.3.

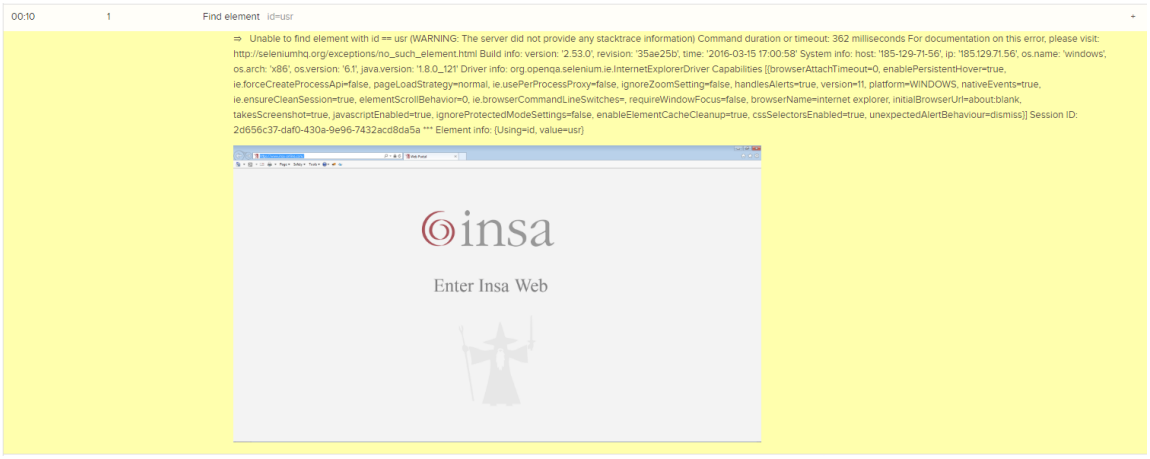


Slika 5.3 Vizualni zapis testa

Tekstualni zapisi vrlo su slični vizualnim zapisima, osim što umjesto snimki zaslona nude tekstualni opis onoga što se događa u testu. Na slici 5.4 slijedi tekstualni zapis istih akcija koje su prikazane na slici 5.3.

Slika 5.4 Tekstualni zapis testa

Pritiskom na znak + s desne strane svakog zapisa dostupne su i proširene informacije o pojedinom koraku. Budući da je svrha alata za otkrivanje pogrešaka upravo otkrivanje pogrešaka, na slici 5.5 je moguće vidjeti kako Browserstack prijavljuje detektiranu pogrešku.



Slika 5.5 Prijavak pogreške u testu

U slučaju sa slike 5.5 dogodila se pogreška prilikom lociranja HTML elementa. Element je tražen preko vlastitog ID-a i nije pronađen, što je Browserstack prijavio na prikladan način. Ovakav način prijavljivanja pronađene pogreške je jednak za vizualne i tekstualne zapise. Potrebno je i napomenuti da je tekstualni zapis također dostupan u Test Explorer prozoru u programu Visual Studio u formatu u kojem Visual Studio prijavljuje pogreške u kodu. Međutim, format kojeg pruža Browserstack je detaljniji, pregledniji i potpuniji, stoga je prikladnije koristiti njegov ugrađeni alat za otkrivanje pogrešaka.

5.2 Organizacija testova

Opisavši način pokretanja testova na Browserstacku i njegove mogućnosti i dodatke koje pruža testeru, slijedi opis organizacije samih testova. Budući da je InsaWeb složena aplikacija s mnoštvom raznolikih kontrola i elemenata koji se implementiraju, ponašaju i identificiraju na različite načine, zanimljivo je prikazati rukovanje takvim elementima unutar testova. Također, određeni elementi različito reagiraju na postavke preglednika na kojemu je pokrenuta aplikacija, stoga je potrebno prikazati rezultate istog testa na drugačijim preglednicima. Testiranje mobilnih aplikacija se zbog određenih prepreka u pisanju testova odvija ručno. Nakon određene količine vremena provedenog u proučavanju rukovanja naredbama karakterističnim za osjetljive ekrane mobilnih uređaja, donesen je zaključak kako metode koje pruža WebDriver u jeziku C# nisu dovoljno prilagođene potrebama testiranja spomenutih uređaja. Iako glavni fokus ovog rada nije ručno testiranje, zbog raznolikosti završnih rezultata je dobro spomenuti i analizirati ishode i takvog vida testiranja.

Prema izvoru [browser statistika], prepoznati su web preglednici na kojima je potrebno izvršiti testove. Kao što stoji u izvoru, prva tri preglednika po popularnosti su Chrome, Firefox te Internet Explorer. Podaci koji su uzeti u obzir su važeći za 2016. godinu, budući da su testovi pisani početkom 2017. godine, međutim, značajniji pomak u popularnosti nije se dogodio tijekom proteklih mjeseci tekuće godine. Također, kroz izravnu komunikaciju s korisnicima aplikacije InsaWeb, donesen je zaključak da većina klijenata koji koriste preglednik Internet Explorer ujedno koriste i operacijski sustav Windows 7. Zbog toga je svaki test koji se izvodi na Internet Exploreru pokrenut i na Windows 7. Na sličan način je, putem komunikacije s klijentima, odlučeno da testovi na mobilnim uređajima budu pokrenuti na najnovijim inačicama Android i iOS uređaja. U testovi koji će biti razmatrani prikazani su samo neki od njih. Tablice 5.1 i 5.2 prikazuju strukturu analize testova koje slijede u radu.

Tablica 5.1 Testovi mobilnih uređaja

Mobilni uređaji				
Android			iOS	
Pixel	Nexus 7	Galaxy S7	iPhone 7 Plus	iPhone 7 Plus (zakrenuti ekran)
Prijava na stranicu	Prijava na stranicu		Dijagram vrijednosnica	Dijagram vrijednosnica
Provjera tablice portfelja bez filtera		Provjera tablice portfelja bez filtera	Provjera tablice portfelja bez filtera	

Tablica 5.2 Testovi preglednika na operacijskom sustavu Windows

Operacijski sustavi		
Internet Explorer 11 (Windows 7)	Firefox 49 (Windows 10)	Chrome 54 (Windows 54)
Pregled početne stranice	Pregled početne stranice	Pregled početne stranice
Rukovanje odabirom pogrešnog datuma	Rukovanje odabirom pogrešnog datuma	Rukovanje odabirom pogrešnog datuma
Odabir vrijednosnice	Odabir vrijednosnice	Odabir vrijednosnice

5.2.1 Dokumentiranje testova

Neizbježan segment svakog procesa testiranja jest precizno i uredno dokumentiranje svakog testa. Budući da je cilj svakog testera pronaći pogreške nastale pri izradi programskog

rješenja, logično je da će se isti testovi ponoviti nakon što razvojni programer prijavi otklanjanje pogreške, kako bi se moglo utvrditi da je problem zaista otklonjen. Pri složenim aplikacijama vrlo je teško pamtiti svaku pronađenu pogrešku i svaki scenarij kojega je potrebno izvesti. Zbog toga je nužno provoditi dokumentiranje svakog testa. Svaka dokumentacija testa mora sadržavati osnovne informacije o testu, kao što su scenarij, datum izvršavanja te informacija o prolasku ili padu testa. U dogovoru s razvojnim programerima i dizajnerom koji rade na razvoju i izgledu InsaWeba, složena je tablica za dokumentiranje testova s prikladnim stupcima. U tablici 5.3 moguće je promotriti zaglavlje te tablice.

Tablica 5.3 Zaglavlje tablice za dokumentaciju testova

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Očekivani rezultat	Stvarni rezultati	Prolaz / pad

Prvi stupac tablice za dokumentaciju testova sadržava informaciju o platformi, odnosno, o inačici operacijskog sustava te o pregledniku na kojemu se izvršava test. Sljedeći podatak je datum izvršavanja testa. Nakon njega slijedi stupac u kojemu je definirano ime testa, a nakon njega je stupac u kojemu je pobliže opisan testni slučaj, iako taj opis u praksi ne treba biti suviše opširan. Usporede li se ova dva stupca s podjelom testova na Browserstacku, može se govoriti stupcu u kojemu je definiran *build* i stupcu u kojemu je definiran pojedinačni test. Korisno je da više testnih slučajeva bude organizirano u jedan test. Primjerice, jedan test može biti provjera mehanizma prijave na glavnu stranicu aplikacije, a testni slučajevi mogu pokriti prijavu s točnim podacima, zatim prijavu s netočnim podacima. Testni podaci odnose se na podatke koje tester nužno koristi prilikom testiranja. Najčešće su to korisničko ime i lozinka ili drugi slični osobni podaci koji se unose u tekstualna polja. Ishodi testa uspoređuju se s očekivanim ponašanjem aplikacije, stoga je bitno zabilježiti takvo ponašanje, bilo u pisanom obliku ili u slikovnom prikazu. Na kraju se, prema usporedbi očekivanih i stvarnih rezultata, ocjenjuje prolaz ili pad testa. Ta kategorija se označava bojom. Prikladno, zelena boja označava prolaz, crvena označava pad, žuta boja označava nesigurnost u vezi pada ili prolaska testa nastala zbog nedovoljno informacija, dok se narančastom bojom bilježe testovi koji su prije nekog vremena bili neuspjeli, ali su otklanjanjem pogreške postali uspješni. Odluka o tom je li test zadovoljio određene kriterije tester donosi u suradnji s razvojnim programerom i dizajnerom korisničkog sučelja.

5.2.2 Testovi na mobilnim uređajima

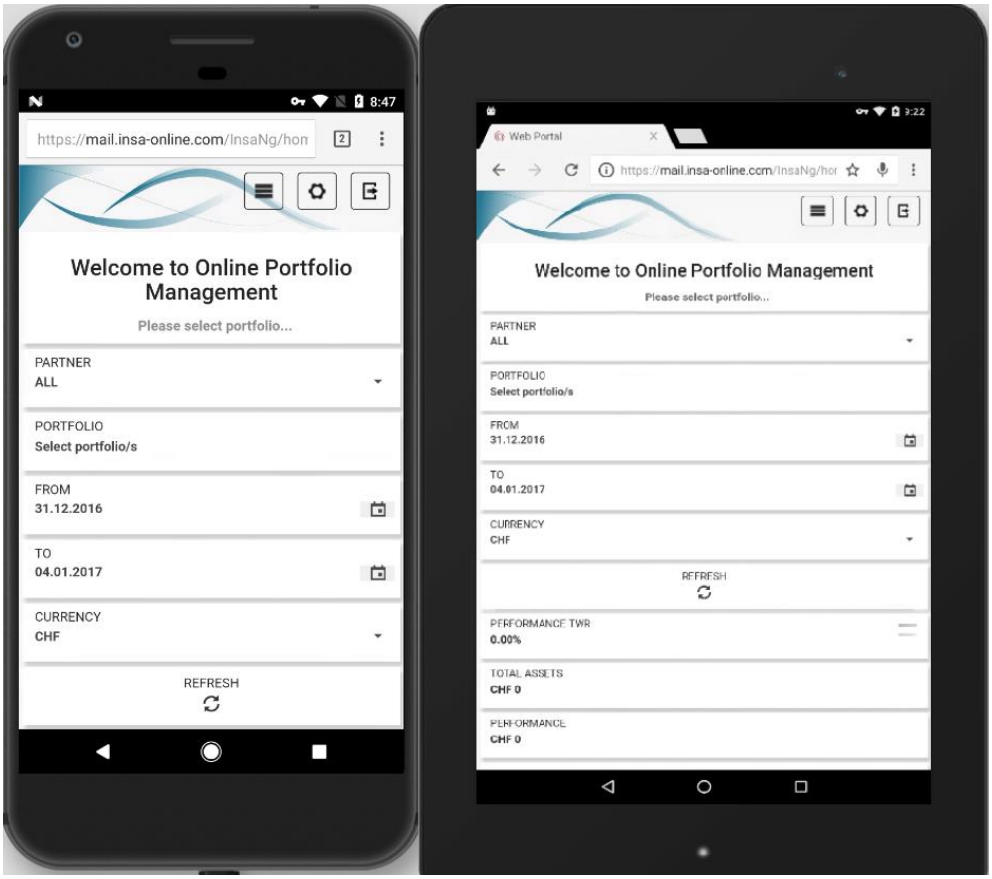
Vodeći se tablicom 5.1 slijedi pregled testova izvedenih ručno na mobilnim uređajima. Cilj je ovih testova prikazati raznolikost problematike pokretanja iste aplikacije na različitim uređajima istog operacijskog sustava i različite rezolucije. U tu svrhu odabrana su tri različita testa, koji su izvedeni na dva ili više različita uređaja.

Prvi promatrani test je jednostavni test koji podrazumijeva prijavu na glavnu stranicu koristeći korisničke podatke. Nakon prijave na glavnu stranicu, snimljen je ekran kako bi se ishod testa mogao vrjednovati. Na tablici 5.4 vidljiva je dokumentacija ovog testa, koji je izveden na uređajima Pixel i Nexus 7.

Tablica 5.4 Test „Prijava“

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Prolaz / pad
Pixel Android	04.01.2017	Prijava	Provjeri stranicu nakon prijave korisničkim podacima	1. Pokreni aplikaciju 2. Unesi korisničke podatke 3. Klikni na „Submit“ gumb 4. Snimi zaslon	Korisničko ime: igor Lozinka: insa	
Nexus 7 Android	04.01.2017	Prijava	Provjeri stranicu nakon prijave korisničkim podacima	-	Korisničko ime: igor Lozinka: insa	

Stupac očekivanih rezultata je u dogovoru testera s razvojnim timom izostavljen pri testiranju mobilnih uređaja, dok će stvarni rezultati biti prikazani u slikovnom formatu na slici 5.6. Koraci testa jednaki su za svaki uređaj pa je zbog nepotrebnog ponavljanja i uštede prostora taj stupac popunjen samo u prvom retku tablice.



Slika 5.6 Test „Prijava“

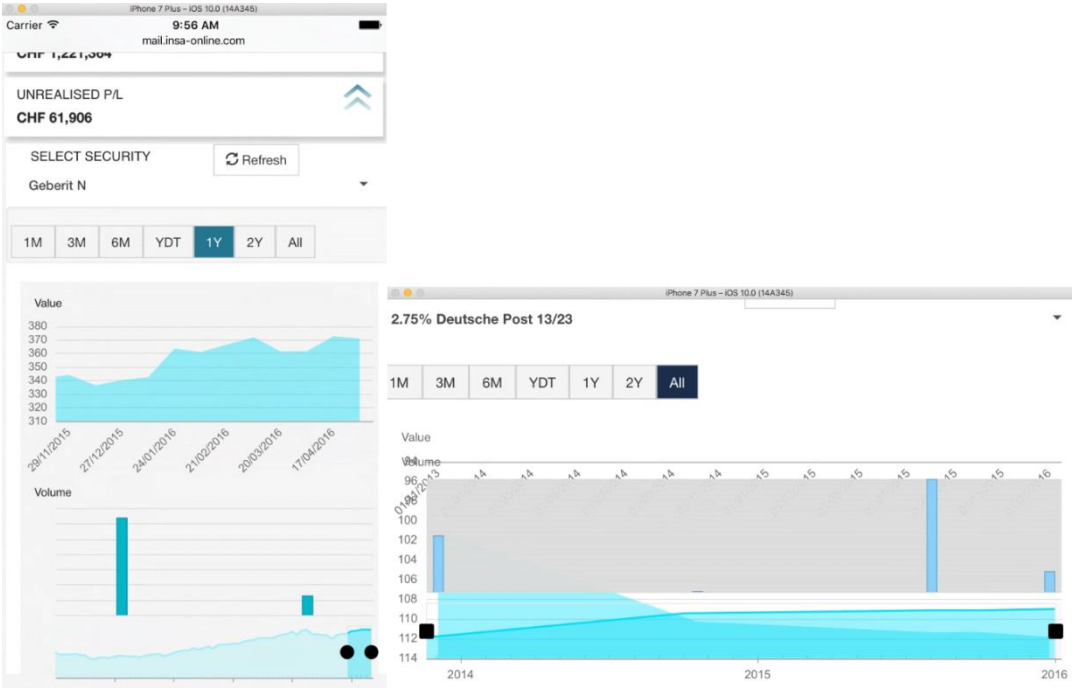
Na slici 5.6 vidljivo je kako nakon prijave na glavnu stranicu aplikacije InsaWeb u slučaju uređaja Nexus 7 tekst na gumbu za osvježavanje stranice je skraćen, odnosno, donji dio svakog znaka nedostaje, dok je prikaz na Pixelu prema procjeni u potpunosti ispravan. Zbog toga je on označen kao test koji je zadovoljio, dok drugi nije.

Sljedeći test je test u kojemu se provjerava dijagram vrijednosnica. Radi raznolikosti su prikazani rezultati toga testa provedenog na iOS uređaju iPhone 7 Plus na uspravnom i zakrenutom ekranu. Slijedi dokumentacija toga testa na tablici 5.5.

Tablica 5.5 Test „Vrijednosnice“

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Prolaz / pad
iPhone 7 Plus Safari 10	28.12.2016	Vrijednosnice	Provjeri dijagram za odabranu vrjiednosnicu	1. Pokreni aplikaciju 2. Unesi korisničke podatke 3. Klikni na „Submit“ gumb 4. Klikni na poveznicu „Security“ 5. Snimi zaslon	Korisničko ime: igor Lozinka: insa	
iPhone 7 Plus Safari 10 (zakrenuti ekran)	03.01.2017	Vrijednosnice	Provjeri dijagram za odabranu vrijednosnicu	-	Korisničko ime: igor Lozinka: insa	

Na slici 5.7 slijede rezultati testa.



Slika 5.7 Rezultati testa „Vrijednosnice“

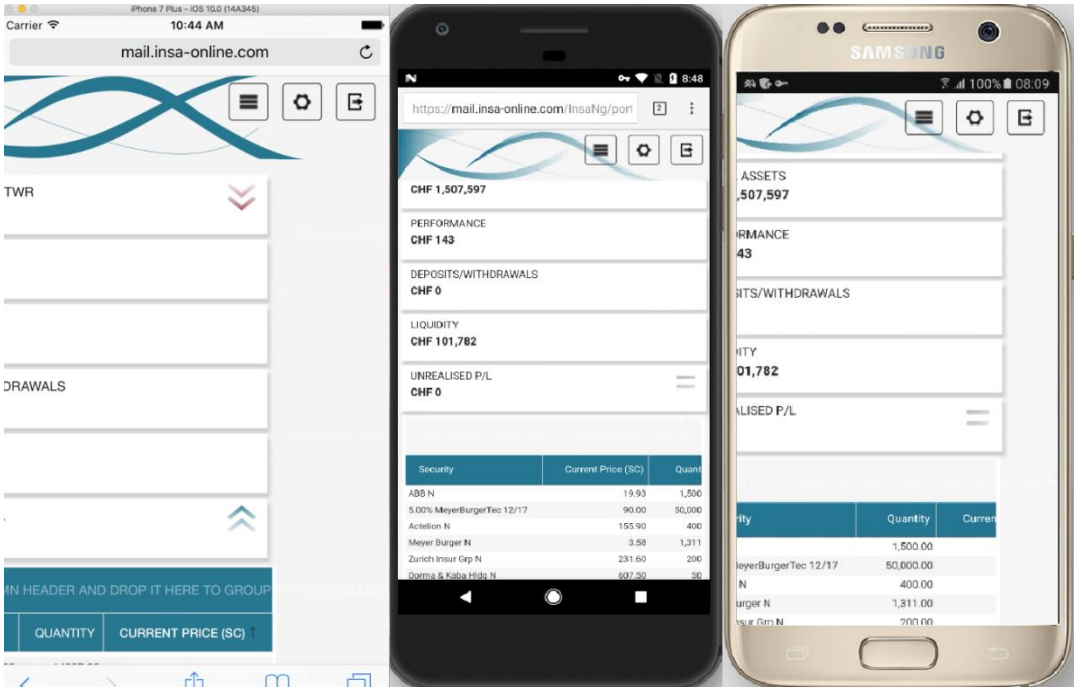
Ovaj je test odabran za prikaz u ovom radu iz razloga što prikazuje važnost rezolucije ekrana za prikaz podataka. U slučaju uspravnog ekrana prikaz dijagrama je ispravan, no promjenom visine i širine ekrana, prikaz dijagrama se raspadne, pri čemu je test pronašao pogrešku u dizajnu aplikacije.

Posljednji test u sklopu testova mobilnih uređaja koji će biti razmatran u ovome radu je test koji obuhvaća dva uređaja Android te jedan uređaj iOS. U testu je provjerena ispravnost rada tablice s filterima u zaglavlju. Filteri se mogu dovlačiti i ispuštati na zaglavlje te organizirati po prioritetima.

Tablica 5.6 Test „Portfelj“

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Prolaz / pad
iPhone 7 Plus Safari 10	27.12.2016	Portfelj	Provjeri portfelj bez filtera	1. Pokreni aplikaciju 2. Unesi korisničke podatke 3. Klikni na „Submit“ gumb 4. Odaberi partnera 1005002 5. Klikni na gumb „Refresh“ 6. Klikni na poveznicu Portfolio 7. Snimi zaslon	Korisničko ime: igor Lozinka: insa	
Pixel Android	04.01.2017	Portfelj	Provjeri portfelj bez filtera	-	Korisničko ime: igor Lozinka: insa	
Galaxy S7 Android	04.01.2017	Portfelj	Provjeri portfelj bez filtera	-	Korisničko ime: igor Lozinka: insa	

Na sljedećoj je slici prikazan ishod testova jednakim redoslijedom kojim su poredani uređaji u tablici 5.6.



Slika 5.8 Rezultati testa „Portfelj“

Sa slike 5.8 moguće je zaključiti kako se jedino Pixel nije suočio s problemom s kojim su se suočili Galaxy S7 i iPhone 7 Plus, a to je izlazak zaglavnog elementa s gumbima iz granica prikaza.

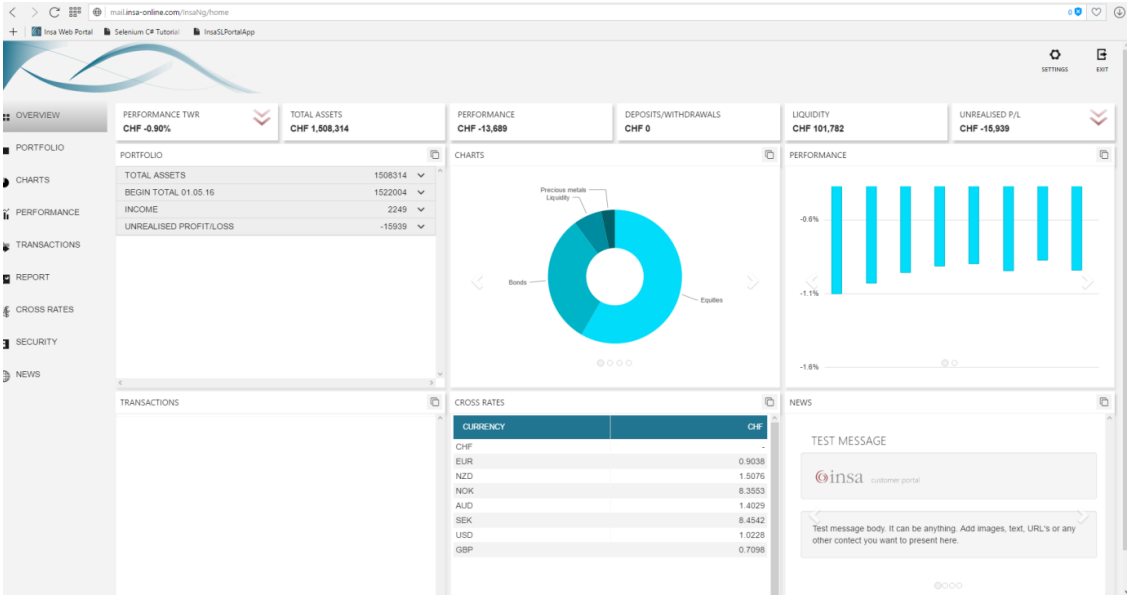
5.2.3 Testovi na operacijskom sustavu Windows

Testovi koji će biti obrađeni u nastavku su automatizirani, odnosno, za razliku od testova na mobilnim uređajima, koraci testnog scenarija su prethodno definirani programskim kodom i ne moraju se izvršavati ručno. Cilj je prikazati testove koji daju jednak rezultat a pokrenuti su na različitim preglednicima, bilo da im je rezultat prolaz ili pad te one koji u slučaju jednakog scenarija na jednom pregledniku ne pronalaze pogrešku dok na drugom pregledniku pronalaze.

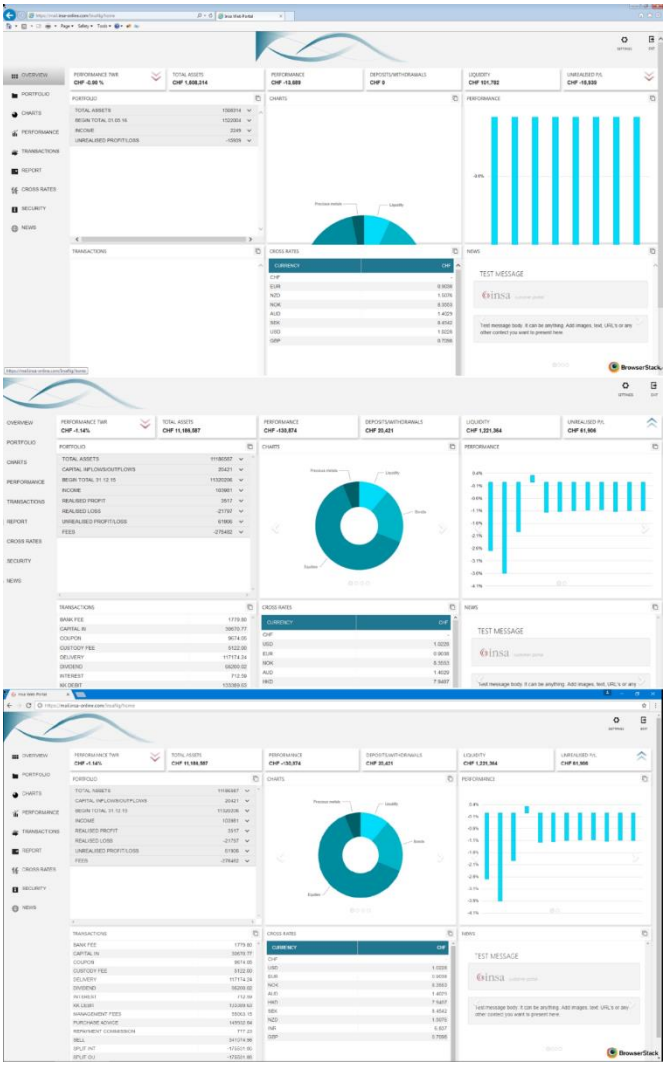
Prvi test koji je razmatran odnosi se na vizualni pregled glavne stranice nakon učitavanja podataka za određenog partnera. Dokumentacija testa nalazi se u tablici 5.7, dok je na slici 5.9 snimka zaslona glavne stranice s učitanim podacima za određenog partnera, koja služi kao referentna slika za pronalazak pogreške u rasporedu i izgledu elemenata stranice. Slika 5.10 nudi usporedbu sva tri preglednika koja se nalaze u dokumentaciji, redoslijedom i kojim se pojavljuju u tablici 5.7.

Tablica 5.7 Test „Glavna stranica“

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Prolaz / pad
Windows 7 Internet Explorer 11	08.12.2016	Glavna stranica	Snimi zaslon glavne stranice	1. Pokreni aplikaciju 2. Unesi korisničke podatke 3. Klikni na „Submit“ gumb 4. Odaberi partnera 1005002 5. Klikni na gumb „Refresh“ 6. Snimi zaslon	Korisničko ime: igor Lozinka: insa	
Windows 10 Firefox 49	21.12.2016	Glavna stranica	Snimi zaslon glavne stranice	-	Korisničko ime: igor Lozinka: insa	
Windows 10 Chrome 54	21.12.2016	Glavna stranica	Snimi zaslon glavne stranice	-	Korisničko ime: igor Lozinka: insa	



Slika 5.9 Očekivani rezultat testa „Glavna stranica“



Slika 5.10 Stvarni rezultati testa „Glavna stranica“

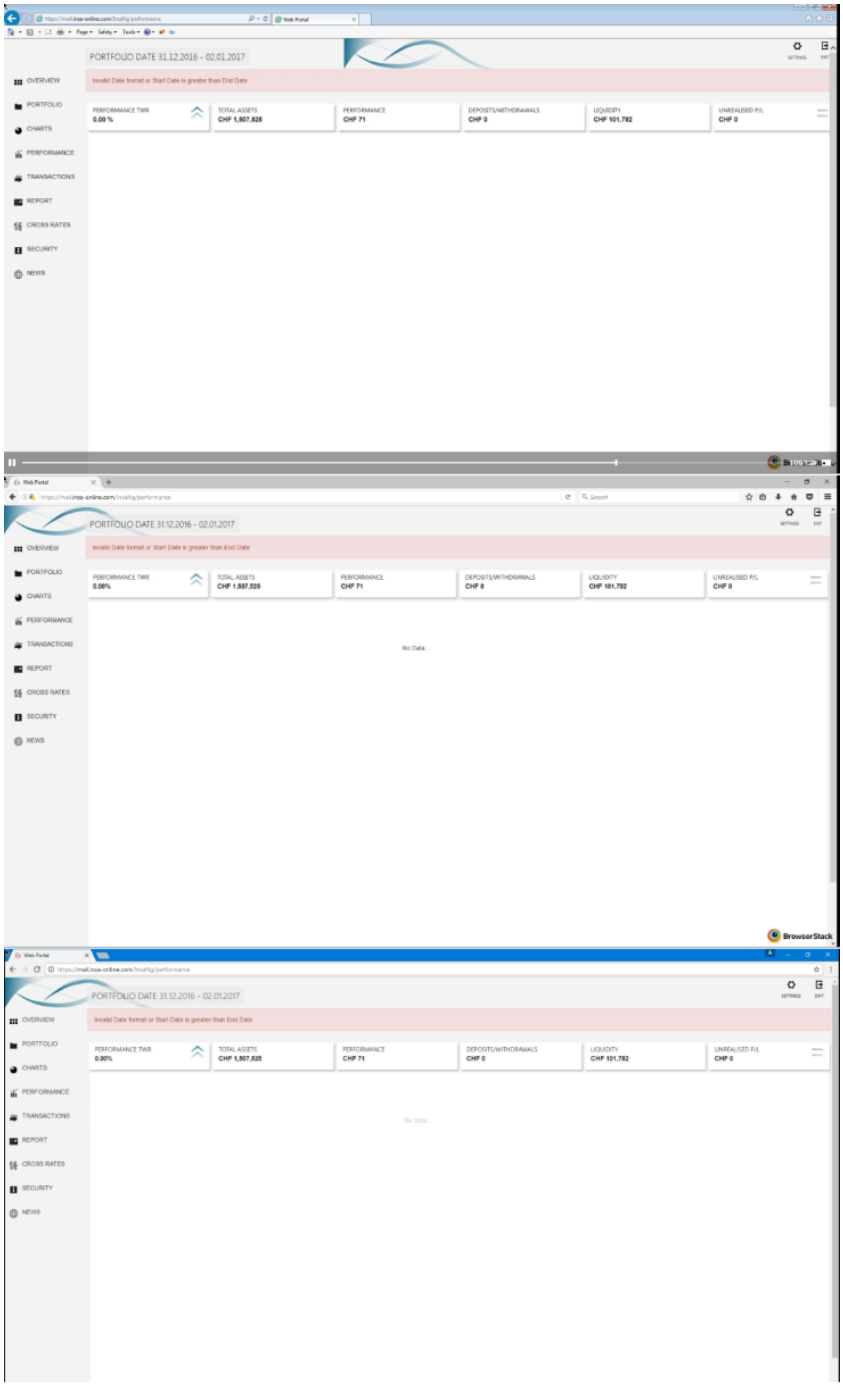
Na temelju slike 5.10 vidljivo je kako Firefox i Chrome zadovoljavaju kriterij izgleda glavne stranice postavljen na slici 5.9, dok Internet Explorer ne zadovoljava. Slika u zaglavlju nije razvučena preko cijelog zaglavlja te SVG dijagrami izlaze izvan okvira svoga spremnika. Pronađena pogreška proslijeđena je dizajneru korisničkog sučelja.

Sljedeći test je test ugrađene funkcionalnosti za ispravljanje neispravnog rukovanja aplikacijom. Ciljani element je kontrola za odabir početnog datuma u čije je polje za unos datuma moguće unijeti bilo koji datum. Poradi toga bitno je obavijestiti korisnika ukoliko pokušava unijeti početni datum koji je veći od završnog, završni datum koji je manji od početnog ili datum za koji ne postoje podaci. Test, čija je dokumentacija prikazana u tablici 5.8, služi ispitivanju postojanja mehanizma koji obavještava korisnika o neispravnom unosu datuma. Implementacija takvog mehanizma je prepuštena razvojnom programeru, stoga ne postoji definirani očekivani rezultat.

Tablica 5.8 Test „Pogrešan datum“

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Prolaz / pad
Windows 7 Internet Explorer 11	02.01.2017	Pogrešan datum	Unesi početni datum za kojeg ne postoje podaci	1. Pokreni aplikaciju 2. Unesi korisničke podatke 3. Klikni na „Submit“ gumb 4. Odaberi partnera 1005002 5. Unesi početni datum 6. Klikni na gumb „Refresh“ 6. Snimi zaslon	Korisničko ime: igor Lozinka: insa Početni datum: 01.04.1993	
Windows 10 Firefox 49	02.01.2017	Pogrešan datum	Unesi početni datum za kojeg ne postoje podaci	-	Korisničko ime: igor Lozinka: insa Početni datum: 01.04.1993	
Windows 10 Chrome 54	02.01.2017	Pogrešan datum	Unesi početni datum za kojeg ne postoje podaci	-	Korisničko ime: igor Lozinka: insa Početni datum: 01.04.1993	

Rezultati testa prikazani su na slici 5.11.



Slika 5.11 Stvarni rezultati testa „Pogrešan datum“

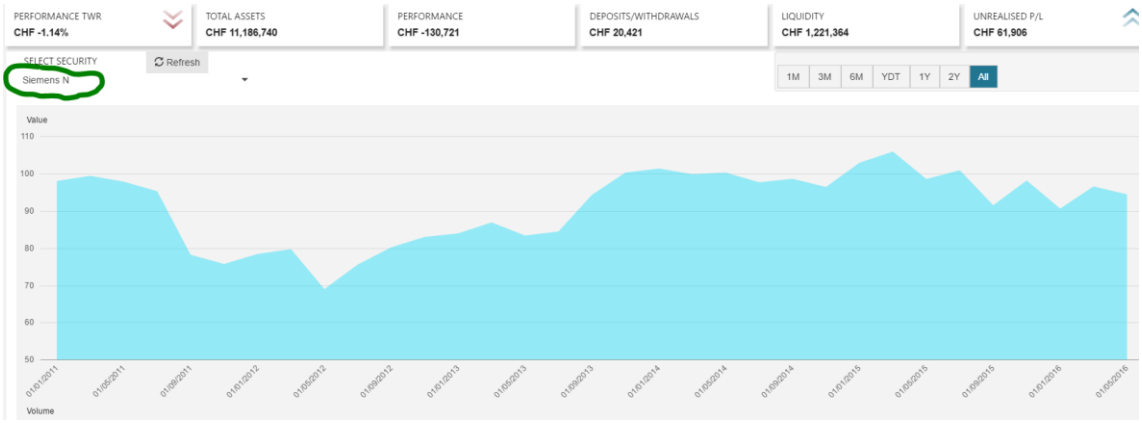
Iz rezultata testa je moguće vidjeti kako sigurnosni mehanizam za rukovanje unosom pogrešnog datuma postoji u obliku obavijesti korisniku i onemogućavanju učitavanja podataka dok se ne unese ispravan datum.

Posljednji test služi za ispitivanje slučaja u kojemu je pronađena pogreška na sva tri preglednika. U tablici portfelja nalazi se popis vrijednosnica, a pokraj imena vrijednosnice se nalazi i gumb, koji u slučaju pritiska vodi izravno na određeni dijagram. Test je dokumentiran u tablici 5.9.

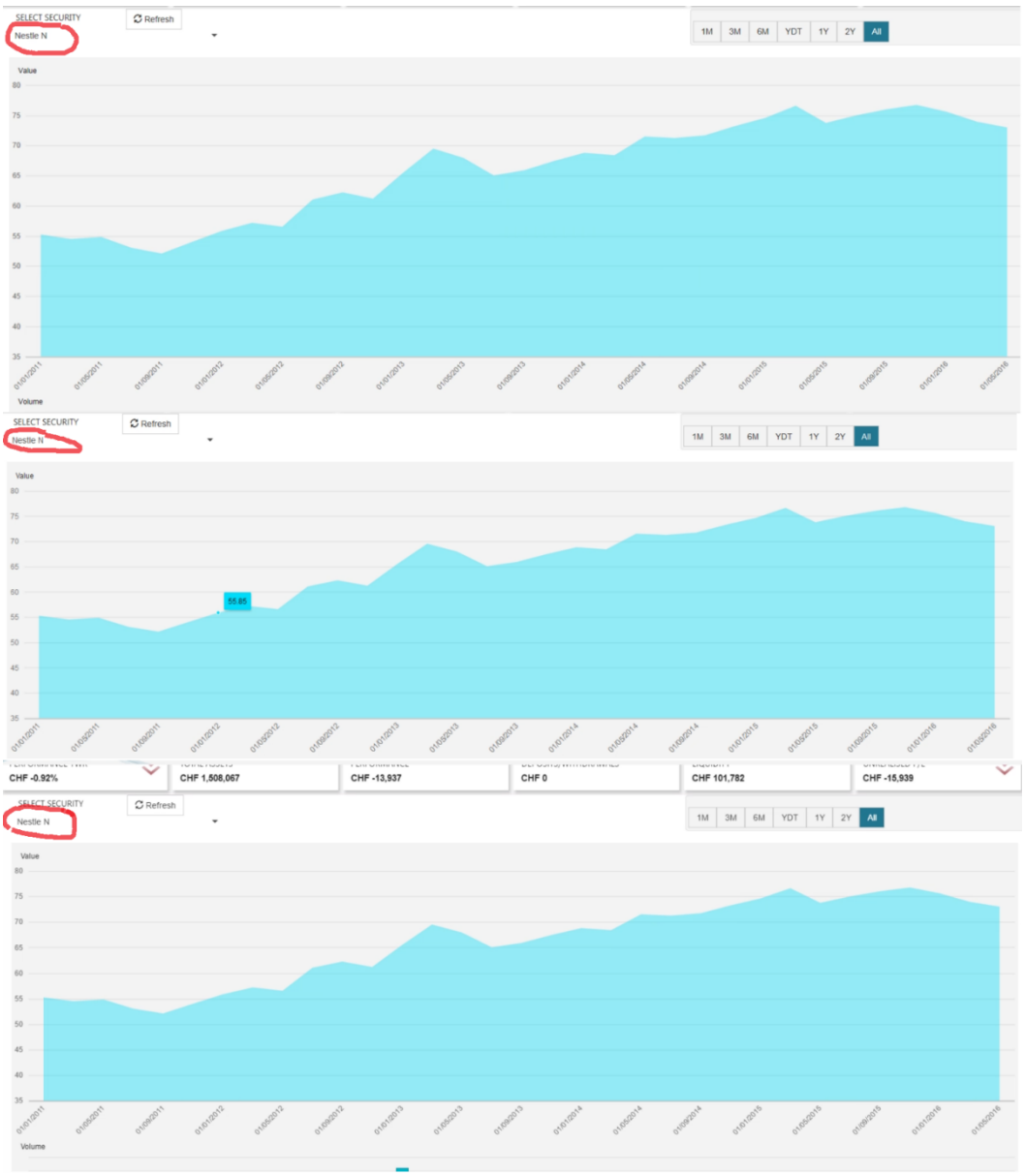
Tablica 5.9 Test „Odabir vrijednosnice“

Platforma - preglednik	Datum	Test	Testni slučaj	Koraci testa	Testni podaci	Prolaz / pad
Windows 7 Internet Explorer 11	22.12.2016	Odabir vrijednosnice	Provjeri je li odabrana vrijednosnica ona za koju se iscrtava dijagram	1. Pokreni aplikaciju 2. Unesi korisničke podatke 3. Klikni na „Submit“ gumb 4. Odaberi partnera 1005002 5. Klikni na gumb „Refresh“ 6. Klikni na poveznicu „Portfolio“ 7. Klikni na redak „Equities“ 8. Klikni na gumb u retku pod nazivom „Siemens“ 9. Snimi zaslon	Korisničko ime: igor Lozinka: insa	
Windows 10 Firefox 49	22.12.2016	Odabir vrijednosnice	Provjeri je li odabrana vrijednosnica ona za koju se iscrtava dijagram	-	Korisničko ime: igor Lozinka: insa	
Windows 10 Chrome 54	22.12.2016	Odabir vrijednosnice	Provjeri je li odabrana vrijednosnica ona za koju se iscrtava dijagram	-	Korisničko ime: igor Lozinka: insa	

Testni slučaj se sastoji od provjere dijagrama vrijednosnice koji daje informaciju o kretanju vrijednosti vrijednosnice tijekom vremena. Iako na glavnoj stranici postoji poveznica „Security“ preko koje se može doći do spomenutog dijagrama te iz padajućeg izbornika odabrati željenu vrijednosnicu, u ovom je testu ispitana mogućnost pristupanja istom dijagramu preko elemenata tablice do koje je moguće doći klikom na poveznicu „Portfolio“. Željeni rezultat i stvarni rezultat dani su na slikama 5.12 i 5.13.



Slika 5.12 Očekivani rezultat testa „Odabir vrijednosnice“



Slika 5.13 Stvarni rezultat testa „Odabir vrijednosnice“

Izvođenjem testa na sva tri preglednika, utvrđena je pogreška koja nije uvjetovana razlikama u pregledniku, već je rezultat propusta prilikom izrade programskog rješenja. Umjesto prikaza dijagrama one vrijednosnice koja je odabrana, prikazuje se dijagram prve vrijednosnice u popisu u padajućem izborniku, označenog crvenom bojom na slici 5.13. Ovoga puta pogreška je prijavljena razvojnom programeru.

Cilj ovoga poglavlja bio je prikazati primjenu testova čija je izrada opisana u prethodnom poglavlju, ali i primjenu ručnog testiranja na mobilnim uređajima. Glavni napor uložen je u odabir prikladnog skupa testova koji prikazuju većinu izazova i kombinacija različitih ishoda koji su proizašli iz testiranja cjelokupne funkcionalnosti aplikacije InsaWeb. Ono što je bitno za naglasiti jest činjenica da se pojedine funkcionalnosti aplikacije različito očituju ako su korištene na različitim uređajima i preglednicima, što je i dokazano priloženim testovima. Iz toga je razloga bitno ispitati svaku moguću kombinaciju korisničkih akcija, pohraniti ih u obliku automatiziranih testnih slučajeva i pokrenuti ih u više okruženja. Posao testera završava izvještajem o ishodu testa ili skupa testova, odakle sav daljnji posao preuzima razvojni programer ili dizajner korisničkog sučelja, ovisno o tome tko je odgovoran za postojanje pronađene pogreške.

6 ZAKLJUČAK

Prema spomenutoj metodologiji rada tvrtke Hammer d.o.o. te složenoj strukturi aplikacije InsaWeb, vidljivo je kako testiranje pojedinih elemenata korisničkog sučelja i programske logike zauzima vitalan dio razvoja programske podrške. Budući da je cilj tvrtke aplikaciju InsaWeb prilagoditi zahtjevima i potrebama klijenata, velik je izazov stvoriti rješenje koje zadovoljava svojstva različitih operacijskih sustava i preglednika te jednako tako i mobilnih uređaja. Iz toga je razloga pomoću skupa alata za testiranje pod nazivom Selenium definiran skup automatiziranih testova napisanih u programskom jeziku C#. Testovi sastavljeni pomoću Seleniuma povezani su s uslugom na oblaku računala pod nazivom Browserstack, koja omogućuje višeplatformsko testiranje na način da pruža simulirane uvjete višestrukih inačica preglednika i operacijskih sustava. Testovi su organizirani prema modelu PageFactory kako bi se postigla ekonomičnost koda i optimalno iskoristili raspoloživi resursi. Korisničko sučelje alata Browserstack korišteno je za praćenje izvršavanja testova kao i za sažimanje rezultata testova. Iz opširnog skupa testova koji su provedeni nad aplikacijom InsaWeb, odabrano je nekoliko primjera koji jasno pokazuju način na koji se ista aplikacija ponaša ako je pokrenuta preko drugačijeg preglednika ili na uređajima čiji su prikazi različite razlučivosti. Reprezentativni testovi su uredno dokumentirani te razlika između njihovog očekivanog i stvarnog rezultata prikazani u obliku snimki zaslona.

Cilj ovog rada bio je ukazati na potrebu za automatizacijom scenarijskih testova te u tom duhu istaknuti njezine prednosti nad ručnim testiranjem. Automatizirani testovi mogu se napisati samo jednom te izvršiti u proizvoljnom broju, ovisno o potrebi. Višeplatformsko testiranje dodatno obogaćuje automatizaciju testova time što proširuje popis uređaja na kojima je moguće provesti testiranje bez nužnog posjedovanja istih uređaja, čime se drastično smanjuju troškovi i vrijeme testiranja.

Proces testiranja funkcionalnosti aplikacije InsaWeb se nastavio i izvan okvira ovog rada te je njegovim doprinosom otkriven značajan broj pogrešaka koje su ispravljene, a klijenti su dobili isporučenu sigurnu i istestiranu inačicu programske podrške.

LITERATURA

[1] Development Cycle Methodology for Insa Portfolio Management System, dokument tvrtke Hammer d.o.o., pristupljeno: ožujak 2017

[2] SDLC – Waterfall Model, https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm, pristupljeno: ožujak 2017

[3] T. Gilb, Evolutionary Delivery versus the „Waterfall Model“, ACM Sigsoft Software Engineering Notes, No. 3 Vol. 10, str. 49-61, srpanj 1985

[4] The Waterfall Model, https://www.technologyuk.net/computing/sad/waterfall-model.shtml, pristupljeno: ožujak 2017

[5] T. Clear, The Waterfall is Dead! Long live the waterfall!!!, The SIGCSE Bulletin, No. 4 Vol. 35, str. 13-14, prosinac 2003

[6] R. S. Pressman, Software Engineering (a practicioner's approach), McGraw Hill, New York, 2005

[7] The Agile Methodology, https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban, pristupljeno: ožujak 2017

[8] J. Bergin, Patterns for Agile Development Practice, ACM, No. 2 Vol. 4, str. 1-14, listopad 2006

[9] SDLC – Agile Model, https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm, pristupljeno: ožujak 2017

[10] Scrum u praksi na ozbiljnim projektima, dokument tvrtke Hammer d.o.o., pristupljeno: ožujak 2017

[11] R.P. Maranzato, M. Neubert, P. Herculano, Moving Back to Scrum and Scaling to Scrum of Scrums in Less than One Year, ACM Sigsoft Software Engineering Notes, str. 125-130, Konferencija OOPSLA '11, listopad 2011

[12] Sauce Labs, https://saucelabs.com, pristupljeno: ožujak 2017

[13] Browsera, http://www.browsera.com/features, pristupljeno: ožujak 107

[14] What is Browserstack?, http://toolscorer.com/listing/browserstack.html, pristupljeno: ožujak 2017

[15] Browserstack, https://www.browserstack.com/, pristupljeno: prosinac 2016

[16] Selenium, http://www.seleniumhq.org/docs/01_introducing_selenium.jsp, pristupljeno: prosinac 2016

[17] Model – View – ViewModel (MVVM) Explained, https://csharperimage.jeremylikness.com/2010/04/model-view-viewmodel-mvvm-explained.html, pristupljeno: svibanj 2017

[18] J. Freeman, J. Järvi, G. Foust, HotDrink: a library for web user interfaces, ACM SIGPLAN Notices, No. 3 Vol. 48, str. 80-83, studeni 2012

[19] The MVVM Pattern, https://msdn.microsoft.com/en-us/library/hh848246.aspx, pristupljeno: svibanj 2017

[20] What is Angular?, https://developer.telerik.com/topics/web-development/what-is-angular/, pristupljeno: svibanj 2017

[21] Angular Architecture Overview, https://angular.io/guide/architecture#!#templates, pristupljeno: svibanj 2017

[22] Dependency Injection Benefits, http://tutorials.jenkov.com/dependency-injection/dependency-injection-benefits.html, pristupljeno: svibanj 2017

[23] What Is REST, https://restfulapi.net/, pristupljeno: svibanj 2017

[24] S. Schreier, Modeling RESTful Applications, ACM, WS-REST '11, str. 15-21, ožujak 2011

[25] Introduction to REST and .net Web API, https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/, pristupljeno: svibanj 2017

[26] What is Web API, http://www.tutorialsteacher.com/webapi/what-is-web-api, pristupljeno: svibanj 2017

[27] IWebDriver Browser Commands in C#, http://toolsqa.com/selenium-webdriver/c-sharp/iwebdriver-browser-commands-in-c-sharp/, pristupljeno: svibanj 2017

[28] PageFactory in C#, http://toolsqa.com/selenium-webdriver/c-sharp/pagefactory-in-c/, pristupljeno: svibanj 2017

[29] Encapsulate Selenium PageObject, http://toolsqa.com/selenium-webdriver/c-sharp/encapsulation-oops-principle/, pristupljeno: svibanj 2017

SAŽETAK

Svrha ovoga rada je osmisliti, napisati i izvršiti skup testova čija je svrha ispitati cjelovitost i ispravnost rada web aplikacije za rukovanje financijskih portfeljima pod nazivom Insa. Praktičan dio rada je proveden u suradnji s tvrtkom Hammer d.o.o. koja je vlasnik aplikacije. U izvedbi praktičnog dijela u obzir se uzima struktura aplikacije te metodologije rada tvrtke. U skladu s time, potrebno je koristiti alat za pisanje i izvršavanje automatiziranih testova koji se mogu ponovno i učestalo koristiti. U tu svrhu korištena je usluga na oblaku računala pod nazivom Browserstack te uzorak za organizaciju programskog koda za testiranje pod nazivom PageFactory. Elementi web stranice s kojima korisnik može upravljati u programskom kodu su predstavljeni kao objekti pomoću skupa alata Selenium, a cjeloviti projekt napisan je u programskom jeziku C#. Provjera rezultata testova izvršena je putem grafičkog sučelja usluge Browserstack, a scenariji provedenih testova i njihovi pripadajući rezultati su organizirani u dokumentaciji testiranja u obliku snimki zaslona te tablica s podacima.

Ključne riječi: sustav za rukovanje portfeljima, metodologija razvoja, testiranje programskog rješenja, automatizirano testiranje, višeplatformsko testiranje

ABSTRACT

MULTIPLATFORM TESTING OF WEB AND MOBILE APPLICATIONS

The main goal of this thesis is to design, write and execute a set of tests whose purpose is to test the integrity and functionality of financial portfolios management application named Insa. The practical part of the thesis was carried out in collaboration with company Hammer d.o.o., which is the owner of the application. While making the practical part, the structure of the application and company development methodology were taken in consideration. Having said that, it is needed to use a tool for writing and executing automated tests which can be reused often. To achieve that, a cloud service named Browserstack and code design pattern for written tests named PageFactory were used. The elements of a web page with which an end user can interact are presented in code as objects using a set of tools called Selenium and the overall project was written in code language C#. The check of test results was carried out through graphical interface of the Browserstack service and scenarios of the executed tests

and their belonging results are organized in test documentation in the form of screenshots and data tables.

Keywords: portfolio management system, development methodology, software testing, automated testing, multiplatform testing

ŽIVOTOPIS

Matija Lekić rođen je 1. travnja 1993. godine u Bjelovaru. Završio je opću gimnaziju u srednjoj školi „August Šenoa“ u Garešnici. Akademske godine 2012./2013. upisao je preddiplomski studij računarstva na Elektrotehničkom fakultetu u Osijeku, današnjem Fakultetu elektrotehnike, računarstva i informacijskih tehnologija. Akademske godine 2014./2015. završava preddiplomski studij računarstva te sljedeće akademske godne upisuje diplomski studij računarstva, smjer Informacijske i podatkovne znanosti. 2016. godine započinje studentsku praksu u tvrtki Hammer d.o.o. te s istom nastavlja suradnju. Od 2017. godine zaposlenik je tvrtke Hammer d.o.o.

PRILOZI (na CD-u)

Prilog 1. Pismeni oblik rada u .doc i .pdf formatu

Prilog 2. Programski kod provedenih testova u obliku projekta alata Visual Studio