SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2229

Sustav za nadgledanu nadogradnju programskih komponenti

Ivan Dujmić

Zagreb, lipanj 2020.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem svojim roditeljima na potpori tokom cijelog studija. Također zahvaljujem mentoru doc. dr. sc. Stjepanu Grošu na izvrsnoj suradnji i pomoći prilikom rada na Diplomskom radu.

SADRŽAJ

1. Uvod

Nadogradnja programske potpore bitan je čimbenik u kontroli sigurnosti informacijskih sustava. Programski proizvodi većinom su predviđeni za dugotrajnu uporabu i zbog toga prolaze kroz različita vremenska i korisnička okruženja. Samim time stvaraju se novi sigurnosni zahtjevi i prijetnje. Širenjem skupa funkcionalnosti povećava se prostor za ranjivosti. Zbog toga treba redovito nadograđivati programske komponente u svrhu prevencije ranjivosti.

Ponekad određene komponente nije moguće nadograditi. Razlog može biti u tome što bi nadogradnja učinila uslugu neispravnom. Zbog toga se organizacije odlučuju na primjenu nenadograđenih, ranjivih komponenti kako bi i dalje pružali zajamčene usluge.

Činjenica da nadogradnja može poremetiti rad programske komponente čini upravljanje nadogradnjom izazovnim poslom. Posljedice kvarova u uslugama programskih komponenata mogu biti zanemarive, ali mogu ići i do granice ljudskih života. Kako bi se moglo uspješno upravljati nadogradnjom programskih komponenti, potrebno je utvrditi radi li komponenta identično prije i poslije nadogradnje. Tako bi se omogućila primjena nadogradnje u izvanrednim situacijama.

U sklopu ovog diplomskog rada razvijen je sustav koji omogućuje paralelno izvođenje dviju različitih verzija aplikacije. Jedna verzija može biti stara, a druga nadograđena. Sustav uspoređuje rad aplikacija te dokumentira razlike i upozorava korisnika na moguće neželjeno ponašanje nadograđene verzije. Rad nadograđene verzije može se smatrati ispravnim ako su razlike svedene isključivo na parametre koji ovise o slučajnim brojevima.

Rad je podijeljen na šest poglavlja. Drugo poglavlje predstavlja teorijskih pregled dodatnih mogućnosti koje bi se mogle ostvariti sa sustavom razvijenim u ovome radu. Treće poglavlja prikazuje arhitekturu sustava. Četvrto poglavlje detaljno objašnjava implementacijski postupak prilikom izrade sustava. Peto poglavlje prolazi kroz postupak ispitivanja sustava na nekolicini aplikacija. Šesto poglavlje iznosi zaključak.

2. Dodatne mogućnosti

U ovom poglavlju prikazani su mogući slučajevi uporabe sustava. Paralelno izvođenje različitih verzija aplikacije može naći primjenu u puno segmenata. Neki od istaknutijih navedeni su u ovom poglavlju. Pošto se sav promet može lako nadzirati i analizirati odmah se nameće primjena u području računalne sigurnosti. Poglavlje prikazuje mogućnosti primjene u području nadogradnji programskih komponenti. Nakon toga iznosi idejne sličnosti sustava s područjem testiranja pod nazivom *Differential testing*. Na kraju se pokazuje mogućnost primjene sustava prilikom prevencije i otkrivanja ranjivosti sustava.

2.1. Nadogradnja programskih komponenti

Nadogradnja programskih komponenti (engl. *software patching*) je skup promjena na računalnom programu u svrhu ažuriranja, poboljšanja ili popravka istoga [2]. Pod nadogradnjom programskih komponenti većinom se smatraju otklanjanja ranjivosti u kodu i općenitih pogrešaka koje mogu biti drugačije prirode (engl. *bugs*).

Povijesno gledano proizvodnja sigurnosnih zakrpa (engl. *patch*) seže još u početke proizvodnje računala i programske potpore. U početcima proizvođači programske potpore su distribuirali sigurnosne zakrpe na papirnoj vrpci ili probušenim karticama. Tako je primatelj mogao izrezati potrebni dio originalne vrpce ta ga zamijeniti sa zakrpnim. Zbog toga i pojam nosi naziv sigurnosna zakrpa. Kasnije se razvojem prijenosnih medija promijenio način prenošenja zakrpa. Primjerice putem CD-ROM-a. Velikim razvojem Interneta nadogradnja je omogućena preuzimanjem zakrpa s Interneta [2].

U današnje vrijeme postalo je konvencionalno da proizvođači programske potpore javno objavljuju nadogradnje svojim korisnicima [3]. Tako se olakšava proces rješavanja pogrešaka te prevencija ranjivosti.

Nadogradnja komponenti radi sigurnosnih razloga je uobičajena pojava u svijetu računarstva. Prije nego što je moguće započeti s nadogradnjom programske kompo-

nente potrebno je pronaći propust u njenoj implementaciji. U najboljem slučaju ranjivost će biti otkrivena prije nego što je proizvod pušten u uporabu te će proizvođač biti obaviješten o tome. Za potrebe otkrivanja ranjivosti većinom se angažiraju timovi stručnjaka u području računalne sigurnosti pa čak i hakeri.

Neovisno o izvoru, proizvođač je dužan proizvesti sigurnosnu zakrpu za pronađenu ranjivosti te osigurati svojim korisnicima sigurno korištenje njihovog proizvoda [1].

Cilj razvoja nadogradnje programske potpore je proizvesti sigurnosnu zakrpu koja se odnosi na točno definiran problem, te tako ne uvesti nove probleme [1]. U praksi se unos pogrešaka u aplikaciju nadogradnjom koda naziva *software regression*. Ravnoteža između sigurnosti aplikacije i njezinih funkcionalnosti je uobičajen problem. Potrebno je ostvariti dovoljnu razinu sigurnosti, a pritom omogućiti da aplikacija izvršava sve svoje uloge ispravno.

Osnovna podjela sigurnosnih zakrpa je podjela na tri tipa zakrpa: Binarne zakrpe, zakrpe izvornim kodom i velike zakrpe [2].

Binarne zakrpe sadržane su u obliku izvršne datoteke. Pokretanjem izvršne datoteke pokreće se program koji upravlja postavljanjem zakrpe na ciljanu aplikaciju koja se nalazi na zadanom uređaju. Ovaj slučaj je uobičajen kada je aplikacija zaštićena autorskim pravima te je pregled njezinog izvornog koda onemogućen. U slučaju da se radi o aplikaciji s otvorenim kodom, binarne zakrpe predstavljat će datoteke sa ispravnim strojnim kodom. Postoje alati koji će izvršiti zamjenu starog koda s novim kodom [2].

Zakrpe u obliku izvornog koda uobičajene su za projekte razvoja aplikacija otvorenog koda. Zakrpe su većinom prikazane u obliku razlika starog i novog koda što se naziva *diffs*. Od korisnika se očekuje da prevede novu verziju koda kako bi mogao koristiti nadograđenu verziju aplikacije [2].

Velike zakrpe predstavljaju velike promjene u aplikaciji. Pošto sama riječ zakrpa asocira na manji popravak potrebno je uvesti pojam koji će definirati velike popravke. Primjena ovog tipa nadogradnji pronalazi se u ažuriranjima operacijskih sustava primjerice *Windows* operacijskih sustava [2].

Osnovne varijante zakrpa koje vrijedi navesti su: *Hotfix*, točkasto izdanje (engl. *Point release*), privremeni popravak (engl. *Program temporary release*), sigurnosne zakrpe (engl. *Security patches*), paket usluga (engl. *Service pack*) i neslužbene zakrpe *Unofficial patch* [2].

Proces razvijanja sigurnosnih zakrpa može u potpunosti biti odrađen ručnim putem na način da je programer prepušten pisanju cjelokupnog koda. Također, moguće je iskoristiti postojeće alate za generiranje sigurnosnih zakrpa [4]. U praksi alati za

detekciju pogrešaka u kodu imaju široku primjenu u području razvoja nadogradnje programske potpore. Primjenom alata postiže se određena razine automatizacije proizvodnje sigurnosnih zakrpa.

Primjerice alati za otkrivanje pogrešaka mogu programerima karakterizirati i lokalizirati dio koda koji se popravlja, dok alati za primjenu zakrpa mogu sistematizirati stvaranje određenih zakrpa koje se mogu primijeniti u identificiranom kontekstu koda [4].

Nakon što je napisan kod za nadogradnju potrebno je zakrpu uobličiti tako da je jednostavno dostupna korisnicima. Distribucija nadogradnje te obavještavanje korisnika o novoj verziji aplikacije bitan je posao prozivođača programske potpore. Od velikog je značaja što brže i efikasnije korisnicima prenijeti nadogradnje. Zbog toga sustavi za distribuciju nadogradnji moraju imati automatiziran postupak prenošenja nadogradnji do krajnjih korisnika. Neki od poznatih načina dohvaćanja nadogradnji sustava su: *Microsoft's Windows Update*, *Apple's Software Update service* i *Red Hat Network* [1].

Prilikom distribuiranja nadogradnje prema korisnicima potrebno je obratiti pažnju na ograničenja u području veličine nadogradnje. Uz to potrebno je imati na umu kapacitet i propusnost sustava koji dostavlja nadogradnju korisnicima. Ako se zanemare ova ograničenja moguće je izvesti *Denial of service* napad na sustav [1].

Sustav izrađen u sklopu ovog rada mogao bi se primijeniti prilikom procesa upravljanja nadogradnjom programskih komponenti. Pošto je omogućeno paralelno izvođenje dviju verzija aplikacije proces upravljanja nadogradnjom programskih komponenti može se uvelike olakšati nadziranjem razlika u radu različitih verzija aplikacija.

Sustav dokumentira rad aplikacija te njihove razlike. Dokumentiranje može uvelike olakšati proces istraživanja grešaka u nadogradnji, odnosno koji dio zakrpe je poremetio dotadašnji ispravan rad aplikacije.

Nakon određenog vremena korištenja aplikacije moguće je naučiti koje su razlike implicitno ugrađene u kod, odnosno koje razlike ovise isključivo o generiranju slučajnih brojeva. Kada se dođe do te faze, tada sustav može potvrditi je li nadogradnja ispravna ili neispravna. Ako nadograđena verzija aplikacije daje iste izlaze (osim dijelova koji ovise o slučajnim brojevima), onda se radi o ispravnoj nadogradnji aplikacije. U protivnom postoji potencijalna pogreška u kodu, a može se raditi i o razlici koja je povezana s generiranjem slučajnih brojeva s kojom se korisnik još nije susreo.

2.2. Diferencijalno testiranje

Diferencijalno testiranje (engl. *differential testing*) je oblik nasumičnog ispitivanja. Nadopunjava progresivno testiranje temeljeno na komercijalnim testovima i lokalno razvijenim testovima tijekom razvoja i uvođenja proizvoda. Diferencijalno testiranje zahtijeva da ispitivač na raspolaganju ima dva ili više usporedivih sustava. Ovi sustavi predstavljeni su iscrpnim nizom mehanički generiranih ispitnih slučajeva. Ako se rezultati razlikuju ili se događaju nepredviđeni ishodi kao beskonačne petlje i rušenje programa, može se reći da ispitivač ima kandidata na kojem može istraživati razlog pogreške [5].

Diferencijalno testiranje rješava problem troška vrednovanja rezultata ispitivanja. Ako se jedan test provede na više usporedivih programa (primjerice C prevoditelja) i jedan program završi s različitim izlazom, onda se može zaključiti da postoji potencijalna pogreška u kodu [5].

Moguće je prikupiti milijune testova. Jedna uočena pogreška među velikim brojem ispitivanja ima veliku vrijednost. Jedna pogreška može značiti propagiranje pogreške kroz više modula unutar programske potpore.

Ponekad je isplativije koristiti brzinu računalnih ciklusa na postojećim testovima te ih koristiti na više usporedivih primjera, nego ulagati ljudski napor za osmišljavanje i ocjenu testova [5].

Fizičari čestica koriste istu paradigmu: ispituju milijune uglavnom dosadnih događaja da bi pronašli nekoliko interakcija čestica s velikim zanimanjem [5].

U praksi se pokazalo da diferencijalno testiranje ima uspješnu uporabu u pronalasku logičkih i semantičkih pogrešaka [6].

Moguća je primjena diferencijalnog testiranja u različitim domenama, primjerice: SSL/TLS implementacije, C prevoditelji, JVM implementacije, Vatrene stijene i anitvirisni programi [6].

Jedan od problema diferencijalnog testiranja je efikasnost testiranja. Potrebno je omogućiti ulaze za testiranja određenog broja usporedivih aplikacija. Uz efikasnost testiranja jedan bitan problem predstavlja pojavljivanje *false positive* slučajeva. U nekim slučajevima razlike ne moraju nužno označiti primjerak programa kao pogrešan [5].

Efikasnost testiranja za sobom povlači kvalitetu ispitnih primjera. Pošto broj testova može biti jako velik, potrebno je pružiti ulaze za sva ispitivanja.

Primjerice ako uzmemo u obzir slučaj C prevoditelja. Moguće je svakom testiranju kao ulaz dati nasumično stvoren tekst. Svaki ispitni primjerak će dati određeni rezultat.

Problem nastaje što nasumično stvoren tekst ne ulazi u srž rada C prevoditelja. Nasumično generirani ispitni primjerci nemaju nikakvu poveznicu o domeni programa te tako mogu davati nekvalitetna rješenja koja ne mogu ukazati na potencijalne pogreške u logici programa [5].

Generiranje ispitnih primjeraka može se podijeliti na generiranje s vodičem (engl. *guided*) i generiranje bez vodiča (engl. *unguided*) [6].

Generiranje ispitnih primjeraka s vodičem uzima u obzir izlaze ispitivanja iz do sada završenih iteracija testiranja. Tako se pokušava minimizirati broj potrebnih ispitnih primjeraka potreban za otkrivanje pojedine pogreške. Generiranje ispitnih primjeraka s vodičem dodatno se može podijeliti na dva tipa: *Domain-specific evolutionary guidance* i *Domain-independent evolutionary guidance* [6].

Domain-specific evolutionary guidance u obzir uzima domenu programa koji se testira. Potrebno je dobro poznavanje pozadine programa zbog relevantnog generiranja ispitnih primjeraka. Primjer jednog takvog sustava za generiranje ispitnih primjeraka je Mucerts [6].

Domain-independent evolutionary guidance ne uzima u obzir poznavanje domene programa koji se ispituje. NEZHA je primjer takvog sustava za generiranja ulaznih primjeraka. Ključna motivacija iza NEZHA-inog dizajna je da trenutni alati generiraju podatke jednostavnim posuđivanjem tehnika dizajniranih za pronalaženje grešaka u slučaju pada sustava ili oštećenja memorije u pojedinim programima primjerice programima za maksimiziranje pokrivanja kodova. Suprotno tome, NEZHA koristi asimetričnosti u ponašanju između više programa testiranja kako bi se usredotočila na ulaze koji s većom vjerojatnošću izazivaju semantičke pogreške [7].

NEZHA je otkrila 778 jedinstvenih, do sada nepoznatih opasnosti u širokom rasponu aplikacija (ELF i XZparsers, PDF preglednici i SSL / TLS knjižnice), od kojih mnoge čine dosad nepoznate kritične sigurnosne ranjivosti. Pomoću NEZHA-e pronađene su razlike u SSL/TLS implementaciji validacije X.509 certifikata. Pronađene razlike kreću se od pogrešnog rukovanja određenim tipovima proširenja do neprimjerenosti posebno izrađenih isteklih certifikata koji omogućuju *man-in-the-middle* napade [7].

Problem *false positive* slučajeva može se također prikazati na primjeru C prevoditelja. Razlika u izlazima ispitivanja može postojati, a da su oba izlaza ispravna. Ponekad C prevoditelj može birati više rješenja za isti problem te se zbog toga javlja više mogućih puteva do ispravnog rješenja. Ovaj problem se može pojaviti u više slučajeva i predstavlja ograničenje koje treba uzeti u obzir prilikom korištenja metode diferencijalnog testiranja [5].

Teorijski gledano, diferencijalno testiranje se zasniva na dvije osnovne ideje. Ako se za primjer uzmu dva sustava: S i S'. Prva ideja glasi: sustav S nema pogrešaka u sebi te je on glavna verzija programa. Samim time primjena diferencijalnog testiranja na glavnoj verziji aplikacije nema smisla ako nije uvedena nova verzija S' u kojoj postoje potencijalni problemi [8].

Druga ideja glasi: kako bi uočili sve razlike između sustava S i S' nije dovoljno testirati sustave pomoću testnog skupa T_S koji karakterizira ponašanje sustava S. Sustav S treba biti uspoređen sa sustavom S' korištenjem testnih skupova T_S i $T_{S'}$ kako bi se u obzir uzeo i testni skup koji karakterizira ponašanje sustava S' [8].

Sličnost diferencijalnog testiranja s idejom ovoga rada je poprilično velika. U oba slučaja testiranje se provodi usporedno na više verzija aplikacije. U slučaju ovoga rada ispitivat će se dvije verzije iste aplikacije. Kod diferencijalnog testiranja to ne mora biti slučaj. Dovoljan uvjet za uspješno diferencijalno testiranje je da su aplikacije usporedive, odnosno da im je princip rada sličan.

2.3. Zaštita od napada

Sustav izrađen u ovom radu može biti iskorišten u svrhu zaštite od potencijalnih napada. Gotovo svaka aplikacija sadrži ranjivosti koje se mogu zlouporabiti za ugrožavanje sustava. Ako pretpostavimo da dvije aplikacije imaju identičnu funkcionalnost, ali ih implementiraju dva različita proizvođača, tada je vjerojatnost da će ranjivost biti na istom mjestu manja.

Ako se dvije verzije aplikacije pokrenu u paraleli i nadzire se njihovo ponašanje moguće je pratiti njihove razlike. Ako se radilo o aplikacijama koje su prije toga radile identično, znači da je sustav kompromitiran i da su u jednu od verzija unesene maliciozne aktivnosti. Odnosno može se reći da je nad aplikacijom izvršen napad.

Na ovaj način olakšava se identifikacija problema u sigurnosti što je bitna stavka u primjeni izrađenog sustava. Nakon identifikacije potrebno je izvršiti analizu samog napada i izvršiti korake koji izlaze iz okvira ovoga rada.

3. Arhitektura sustava

U ovom poglavlju opisana je arhitektura sustava izrađenog u sklopu rada. Ilustrativno je prikazana skica sustava te izneseni osnovni koncepti i ideje vezane uz izgradnju sustava. Prikazane su sve komponente od kojih se sustav sastoji te su prikazane njihove uloge u radu sustava. Također objašnjen je osnovni način korištenja sustava.

Osnovna namjena sustava je riješiti problem nadogradnje sustava koja potencijalno može narušiti njegovo ispravno funkcioniranje. Idejni pristup tome je pokretanje aplikacija u paraleli te usporedba njihovog ponašanja. Usporedba ponašanja može se izvršiti nadgledanjem paralelnog izvršavanja dvaju verzija aplikacija, jedne nadograđene i jedne osnovne, odnosno nenadograđene. Osnovna verzija po pretspotavci radi ispravno i nema nikakvih pogrešaka u njenom izvršavanju. Nadograđena verzija je potencijalno neispravna i njezino izvršavanje je predmet promatranja. Tako je paralelizacija aplikacija prvi korak prilikom dizajniranja ovoga sustava. Potrebno je omogućiti istovremeni rad dvaju aplikacija te interakciju klijenta s tim aplikacijama.

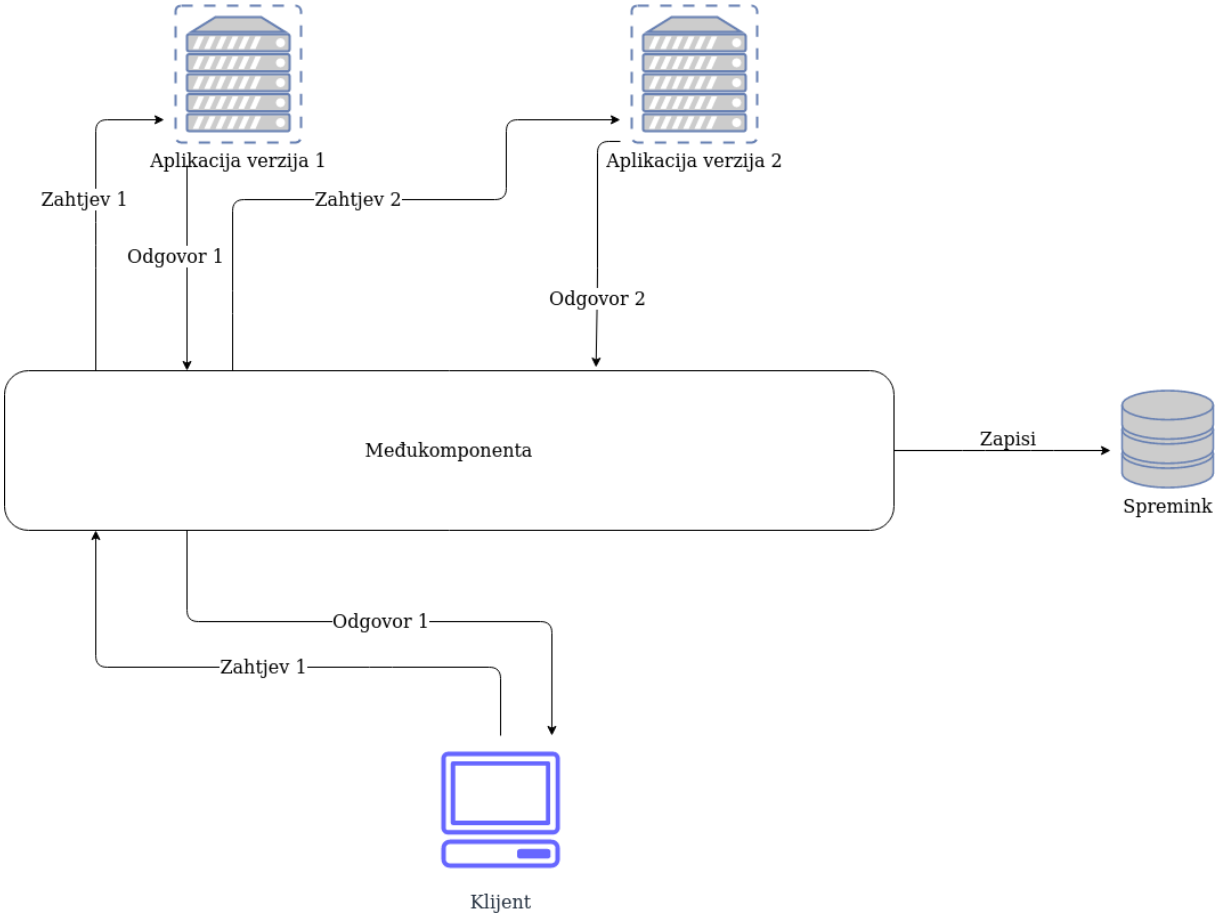
Istovremeno izvršavanje akcija nad dvama aplikacijama nemoguće je izvesti ručnom interakcijom klijenta i aplikacija. U praktičnom smislu klijent može ručno pokrenuti obe verzije aplikacije te činiti iste akcije nad aplikacijama. Idejno to nije cilj. Potrebno je automatizirati korištenje aplikacija na način da korisnik direktno koristi samo jednu verziju aplikacije. Zbog toga sustav treba omogućiti klijentu da vrši interakciju s jednom od verzija dok međukomponenta istovremeno prosljeđuje zahtjeve drugoj verziji aplikacije. Kao drugi korak u izgradnji sustava nameće se potreba za izgradnjom međukomponente koja je zadužena za dohvaćanje klijentskih zahtjeva upućenih jednoj od verzija te prosljeđivanje tih zahtjeva drugoj verziji.

Potrebno je definirati način ocjenjivanja ispravnosti u svrhu ocjenjivanja rada nadograđene verzije aplikacije. Za ocjenjivanje točnosti odabrano je uspoređivanje odgovora aplikacija na razini aplikacijskog sloja. Izgradnja komponente za usporedbu odgovora treći je korak prilikom izgradnje sustava. Potrebna je komponenta koja će primati odgovore od dvije aplikacije. Nakon što je primila odgovore, komponenta treba izvršiti usporedbu odgovora te dokumentirati sve potrebne informacije kako bi

korisnik sustava mogao učiti o razlikama.

Na slici 3.1 prikazana je arhitektura sustava. Klijent komunicira s aplikacijama pomoću međukomponente. Međukomponenta predstavlja drugi i treći korak u izgradnji. Ona je posrednik u komunikaciji i uspoređuje primljene odgovore te ih dokumentira. Zbog toga međukomponenta implementira dvije stvari te ima vlastite potkomponente. Aplikacije su prikazane kao dvije zasebne jedinke. Aplikacije ne smiju u svome radu ovisiti jedna o drugoj. Također aplikacije ne trebaju komunicirati jedna s drugom.

Općeniti scenarij korištenja sustava je sljedeći. Klijent upućuje zahtjev jednoj od verzija aplikacije. Navedeni zahtjev prikazan je kao Zahtjev 1 na slici 3.1. Konkretno, klijent može pokrenuti aplikaciju u web pregledniku te poslati zahtjev za početnom stranicom aplikacije. Međukomponenta će primiti taj zahtjev te će na osnovu njega stvoriti novi zahtjev, Zahtjev 2. Novostvoreni zahtjev bit će identičan zahtjevu za originalnu verziju aplikacije. Jedine razlike bit će implementacijski detalji potrebni da nadograđena aplikacija može procesuirati stvoreni zahtjev. Nakon što su zahtjevi spremni, prosljeđuju se do aplikacija. Aplikacije odgovaraju na zahtjeve i odgovori dolaze do međukomponente. Unutar međukomponente se uspoređuju i pohranjuju. Međukomponenta prosljeđuje Odgovor 1 koji je povezan sa Zahtjev 1. Tako klijent dobiva sadržaj odgovora na poslani zahtjev.



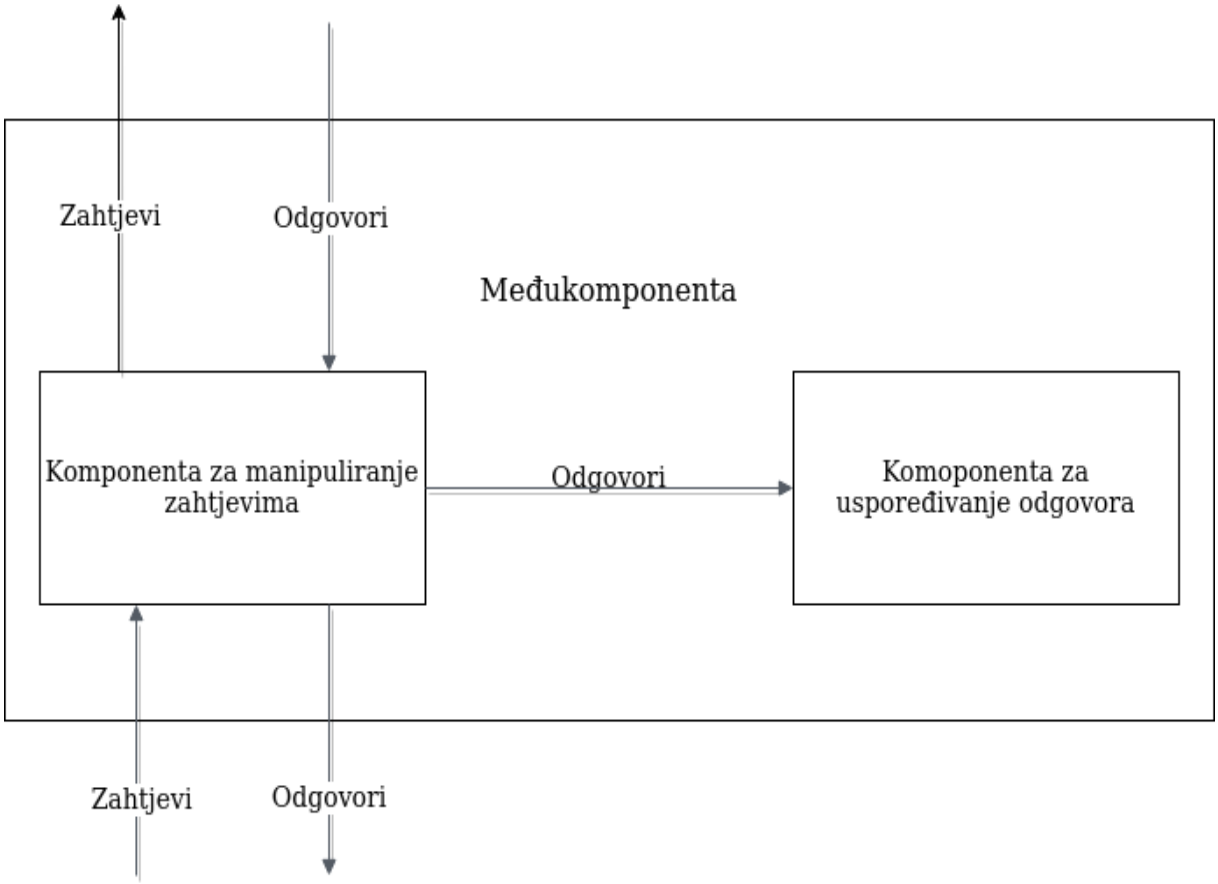
Slika 3.1: Arhitektura sustava

Slika 3.2 prikazuje izgled međukomponente. Međukomponenta je prekompleksna da bi se prikazala jednom jedinicom. S obzirom na to da je zadužena za dvije bitne stavke u radu sustava, potrebno ju je dizajnirati pomoću više potkomponenti. Međukomponenta sastoji se od dvije potkomponente: komponenta za manipuliranje zahtjevima i komponenta za uspoređivanje odgovora.

Komponenta za manipuliranje zahtjevima zadužena je za primanje zahtjeva od klijenta. Od primljenog klijentskog zahtjeva potrebno je stvoriti novi zahtjev za drugu verziju aplikacije. Nakon stvaranja potrebno je zahtjeve proslijediti aplikacijama. Uz prosljeđivanje zahtjeva prema aplikacijama, komponenta za manipuliranje zahtjevima prosljeđuje primljene odgovore na dva odredišta. Parovi odgovora primljeni od aplikacija prosljeđuju se komponenti za uspoređivanje odgovora. Također potrebno je proslijediti jedan od odgovora prema klijentu, ovisno kojoj aplikaciji je klijent poslao zahtjev.

Komponenta za uspoređivanje odgovora prima odgovore koje joj prosljeđuje komponenta za manipuliranje zahtjevima. Primljeni odgovori se uspoređuju i dobiveni

rezultati se dokumentiraju. Klijent ne uspostavlja komunikaciju s komponentom za uspoređivanja odgovora, niti ona komunicira s klijentom.



Slika 3.2: Međukomponenta

4. Implementacija

U ovom poglavlju detaljno je prikazan postupak implementacije sustava. Dani su osnovni koncepti i ideje prilikom izgradnje sustava. Prikazan je pregled osnovnih tehnologija korištenih prilikom implementacije. Uz detaljno objašnjenje prikazani su bitni isječci koda. Poglavlje je podijeljeno u četiri potpoglavlja od kojih svako prolazi kroz potrebne implementacijske postupke.

Postupak implementacije sustava može se postaviti kao postupak rješavanja 3 problema:

- paralelizacija izvođenja aplikacija
- manipulacija zahtjevima
- uspoređivanje odgovora

4.1. Paralelizacija izvođenja aplikacija

Kako bi se omogućilo paralelno izvršavanje akcija nad aplikacijama potrebno je aplikacije odvojiti kao zasebne jedinke. Tako je moguće pristupati svakoj verziji aplikacije kao zasebnoj i neovisnoj jedinci.

Kao rješenje problema nameće se *Docker* tehnologija koja je veoma zastupljena u području razvoja aplikacija, pogotovo web aplikacija.

4.1.1. Docker

Docker je alat dizajniran da olakšava stvaranje, implementaciju i pokretanje aplikacija pomoću spremnika [9]. Docker se pokazuje kao dobro rješenje za uspostavljanje paralelnog izvođenja aplikacija jer pokreće aplikacije kao izolirane spremnike.

Spremnik (engl. *Docker container*) je standardna jedinica softvera koja pakira kod i sve njegove ovisnosti, tako da se aplikacija brzo i pouzdano pokreće iz jednog računalnog okruženja u drugo. Slika spremnika (engl. *Docker container image*) je lagan,

samostalan izvršni paket softvera koji uključuje sve potrebno za pokretanje aplikacije: kod, vrijeme izvođenja, sistemske alate, knjižnice sustava i postavke [10].

Slike spremnika postaju spremnici za vrijeme izvršavanja, a u slučaju Docker spremnika - slike postaju spremnici kada se pokreću na Docker Engineu. Docker će uvijek pokretati isti softver korišten u spremniku, bez obzira na infrastrukturu. Spremnici izdvajaju softver iz svog okruženja i osiguravaju mu ravnomjeran rad, unatoč razlikama, primjerice između razvoja i postupanja [10].

Spremnici i virtualni strojevi imaju slične prednosti izdvajanja i raspodjele resursa, ali djeluju drugačije jer spremnici virtualiziraju operativni sustav umjesto hardvera. Spremnici su prijenosniji i učinkovitiji [10].

Kako bi se omogućilo pokretanje spremnika potrebno je stvoriti sliku spremnika. Sliku spremnika moguće je stvoriti interaktivnim putem korištenjem naredbenog retka. U praksi slike spremnika stvaraju se pisanjem skripte pod nazivom Dockerfile. Za pisanje Dockerfileova korisnici trebaju imati osnovno znanje pisanja shell skripti te osnovno znanje korištenja okruženja distribucijskog softvera Linux.

Pristup stvaranja slike spremnika pisanjem skripte Dockerfile ima mnoge prednosti:

- Slika spremnika može biti vrlo velika (više gigabytea), dok je njen opis Dockerfile mala tekstualna datoteka koja se lako može pohraniti i dijeliti [11].
- Tekstualne datoteke kao Dockerfile idealne su za sustave kao štu je Git, putem kojeg se mogu pratiti eventualne promjene unesene u Dockerfile [11].
- Dockerfile je čitljiv čovjeku. Iz njega se mogu izvući osnovne informacije o korištenim paketima, potrebnom softwareu te varijablama okruženja koji su nužni za pokretanje sustava [11].
- Dockerfile uključuje sve ovisnosti o softveru do razine operacijskog sustava, a izgrađen je pomoću Docker build alata, što čini vrlo malo vjerojatnim da će se rezultat prevođenja slike razlikovati kada se izrađuje na različitim strojevima [11].

U ispisu 4.1 prikazan je primjer izgleda Dockerfilea. Radi se o Dockerfileu jednostavne web aplikacije koja pruža REST API svojim korisnicima. Aplikacija je napisana u ASP .Net Coreu te u prvom retku dohvaća sliku ASP .Net Corea.

Naredba WORKDIR određuje da sve naredne radnje treba poduzeti iz direktorija /app u datotečnom sustavu slike spremnika (nikad datotečni sustav glavnog računala) [12].

Naredba COPY će kopirati označeni direktorij iz datotečnog prostora glavnog računala u navedeni datotečni prostor slike spremnika [12].

Naredba RUN pokreće navedenu naredbu unutar sustava slike spremnika [12].

Naredba EXPOSE informira Docker da slika spremnika sluša na određenim vratima, konkretno vratima 80 [12].

Naredba ENTRYPOINT omogućava konfiguriranje slike spremnika na način da se slika pokreće kao izvršna datoteka [12].

Postoji još naredbi u sintaksi pisanja Dockerfileova, ali to izlazi iz okvira ovoga rada.

.

```
# Get Base Image (Full .NET Core SDK)
FROM mcr.microsoft.com/dotnet/core/sdk:2.2 AS build-env
WORKDIR /app

# Copy csproj and restore
COPY *.csproj ./
RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# Generate runtime image
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2
WORKDIR /app
EXPOSE 80
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "ColourAPI.dll"]
```

Ispis 4.1: Primjer izgleda Dockerfilea

U nekim slučajevima moguće je povezati više Dockerfileova u jednom sustavu. Primjerice moguće je imati Dockerfile koji sadrži sliku aplikacije te Dockerfile koji sadrži opis baze podataka. U tim slučajevima prakticira se stvaranje mreže spremnika te se opis takve mreže piše u docker-compose datotekama. Na slici 4.2 nalazi se

primjer izgleda docker-compose datoteke. Radi se o docker-compose datoteci za istu aplikaciju čiji je Dockerfile opisan na slici 4.1. Pošto je aplikacija opisana u Dockerfile, unutar docker-compose datoteke nalazi se opis baze podataka koja će se povezati s aplikacijom.

Na početku datoteke specificira se verzija docker-composea koja se koristi. Pod services oznakama definiraju se servisi koji će se povezivati u mreži. Oni su u načelu slike Docker spremnika. Prvi takav servis je ms-sql-server koji predstavlja bazu podataka. Definira se njegova slika te se predaju potrebne varijable okruženja. Nakon toga definira se povezivanje vrata računala na kojem se pokreće slika te samog sustava slike spremnika. Drugi servis je opisan u Dockerfileu i nije potrebno ponovno specificirati njegove karakteristike. Jedino je potrebno specificirati koja vrata računala treba povezati s vratima spremnika.

```
version: '3'
services:
  ms-sql-server:
    image: mcr.microsoft.com/mssql/server:2017-latest-
   ubuntu
    environment:
      ACCEPT_EULA: "Y"
      SA_PASSWORD: "Pa55w0rd2019"
      MSSQL_PID: Express
    ports:
    - "1433:1433"
  colour-api:
    build: .
    ports:
      - "5000:80"
```

Ispis 4.2: Primjer izgleda docker-compose datoteke

Iz navedenog primjera REST API sučelja lako je načiniti njegovu kopiju. Dovoljno je promijeniti imena servisa te promijeniti vrata računala koja će se povezati sa vratima slike spremnika. Ako se u docker-compose datoteci promijeni ime servisa sa ms-sql-server na ms-sql-server-1 te vrata sa 1433:1433 na 1434:1433 dobiva se novi spremnik koji predstavlja drugu bazu podataka. Također istu stvar potrebno je napraviti sa servisom colour-api. Tako se dobivaju dvije zasebne aplikacije. Sada je moguće

pokrenuti dvije mreže na različitim vratima te paralelno izvršavati interakciju s obje aplikacije. U jednu aplikaciju moguće je unositi promjene te pratiti razlike. Docker je uvelike pojednostavio proces paralelizacije te su promjene u kodu minimalne kako bi se paralelizacija omogućila.

4.2. Manipulacija zahtjevima

Sustav izrađen u radu promiče ideju istovremenog korištenja više verzija aplikacija na način da klijent direktno komunicira s jednom aplikacijom. Kako bi se postigla automatizacija tog tipa potrebno je u sustav uvrstiti međukomponentu koja je opisana u poglavlju Arhitektura sustava.

Manipulacija zahtjevima jedna je od dvije uloge međukomponente. Ideja korištenja sustava je da korisnik putem nekakvog pristupnog sučelja, primjerice web preglednika, pristupa aplikaciji. Ako korisnik tako komunicira s osnovnom verzijom aplikacije, potrebno je sve poslane zahtjeve dohvatiti, replicirati te ih poslati na osnovnu verziju aplikacije i na nadograđenu verziju aplikacije. Korištenje proxya nameće se kao idealno rješenje za ovakav problem.

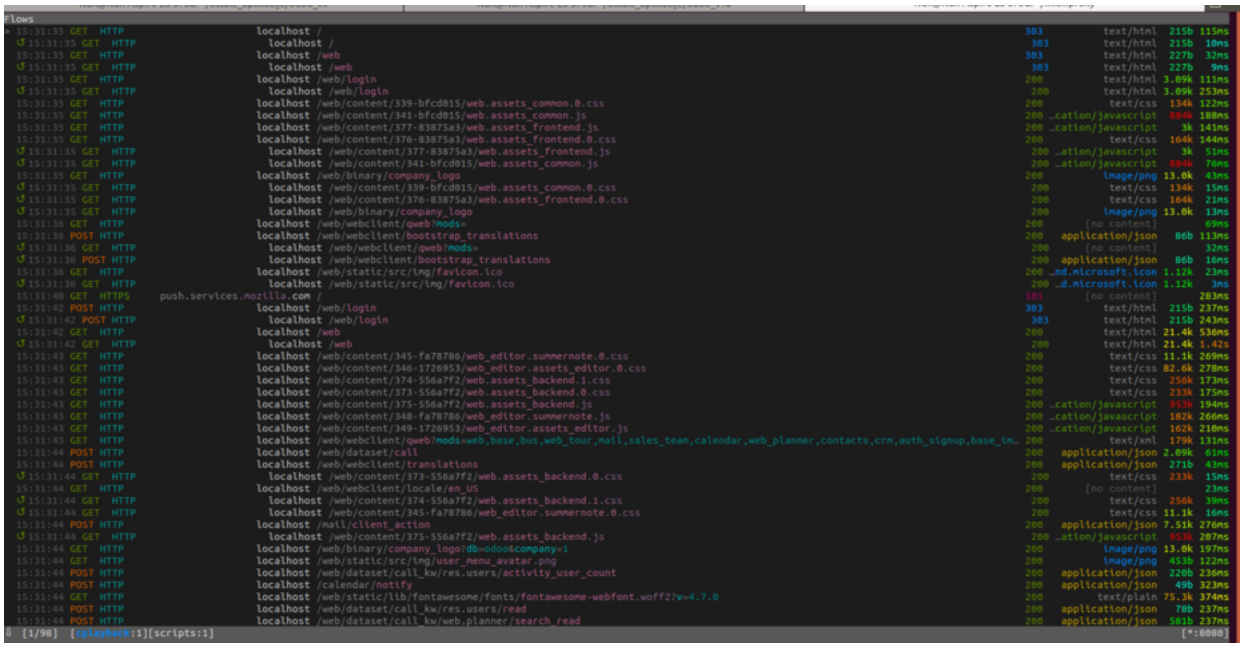
Proxy je posrednički poslužitelj koji razdvaja krajnje korisnike od web lokacija koje pretražuju. Proxy poslužitelji pružaju različite razine funkcionalnosti, sigurnosti i privatnosti, ovisno o slučaju, potrebama ili poslovnoj politici. Internetski promet teče kroz proxy poslužitelj na putu do tražene adrese. Potom se zahtjev vraća preko istog proxy poslužitelja (postoje iznimke od ovog pravila), a zatim proxy poslužitelj prosljeđuje podatke primljene s web mjesta. Suvremeni proxy poslužitelji rade puno više od prosljeđivanja web zahtjeva, a sve u ime sigurnosti podataka i performansi mreže. Proxy poslužitelji djeluju kao vatrozid i web filtar, pružaju zajedničke mrežne veze i predmemoriraju podatke kako bi se ubrzali uobičajeni zahtjevi. Konačno, proxy poslužitelji mogu pružiti visoku razinu privatnosti [13].

Kada bismo postavili Proxy između korisnika i aplikacija koje se koriste, dobili bismo izgled sustava predstavljen na slici 3.1. gdje se unutar međukomponente nalazi proxy. Potrebno je omogućiti stvaranje novih zahtjeva iz jednog postojećeg zahtjeva kojeg je korisnik poslao. Također potrebno je izvršavati promjene nad novostvorenim zahtjevom kako bi ga druga aplikacija mogla prihvatiti. Proxy koji sadrži tražene funkcionalnosti je mitmproxy te je on iskorišten za implementaciju komponente za manipulaciju zahtjevima.

4.2.1. Mitmproxy

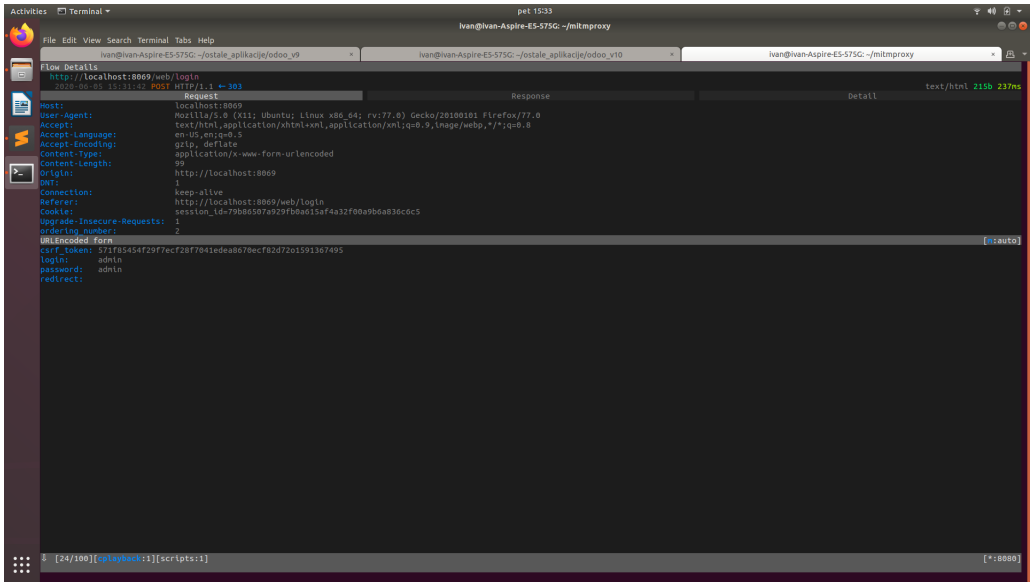
Mitmproxy je interaktivni, SSL / TLS probojni proxy. Može se koristiti za presretanje, pregled, izmjenu i ponovno reprodukciju web prometa poput HTTP / 1, HTTP / 2, WebSockets ili bilo kojeg drugog protokola zaštićenog SSL / TLS. Pomoću mitmproxya moguće je dekodirati razne vrste poruka u rasponu od HTML-a do Protobufa, presresti određene poruke u letu, mijenjati ih prije nego što stignu na odredište i kasnije ih ponovo reproducirati na klijentu ili poslužitelju [14].

Mitmproxy pruža grafičko sučelje putem kojega se lako može pregledati sadržaj prometa dohvaćenog od strane proxya. Na slici 4.1. prikazan je izgled grafičkog sučelja alata mitmproxy.



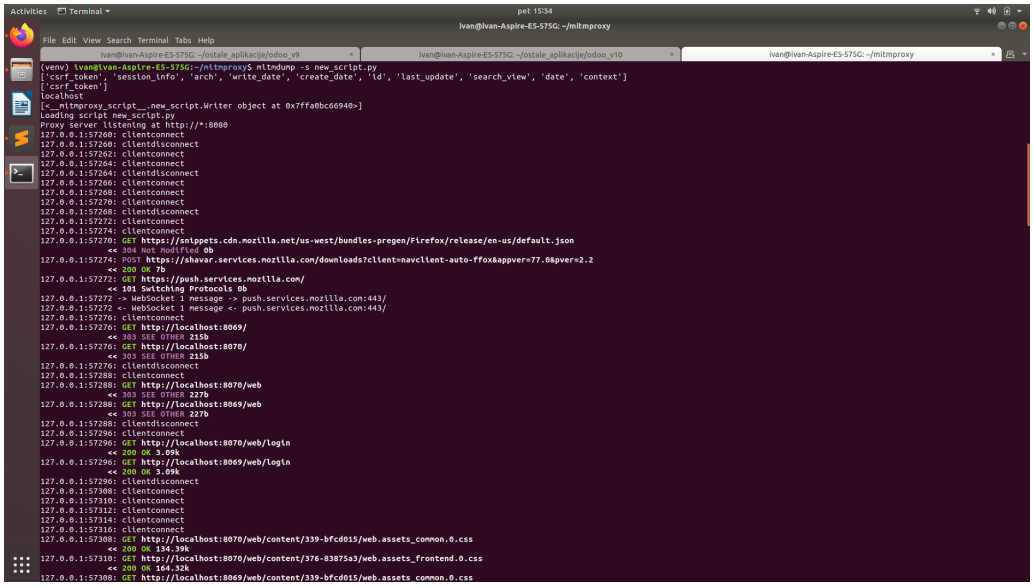
Slika 4.1: Izgled grafičkog sučelja alata mitmproxy

Pomoću alata mitmproxy moguće je otkriti više detalja o svakom pojedinom zahtjevu i odgovoru. Na slici 4.2. nalazi se prikaz detalja o pojedinačnom zahtjevu.



Slika 4.2: Prikaz pojedinačnog zahtjeva

U paketu uz mitmproxy dolazi i alat mitmdump. Mitmdump je verzija mitmproxya bez grafičkog sučelja. Može se iskoristiti za debugiranje proxy skripti pošto se sav ispis nalazi u naredbenom retku. Na slici 4.3 prikazan je izgled sučelja alata mitmdump.



Slika 4.3: Izgled alata mitmdump

Mitmproxy pruža moćan python API koji je iskorišten za pisanje proxy skripte ovoga rada.

Mitmproxy skripta sastoji se od karakterističnih proxy funkcija. Postoji velik broj funkcija koje korisnik može koristiti. U ovome radu korištene su sljedeće: request i

response. Navedene funkcije moguće je napisati unutar klase, ako će postojati određene članske varijable koje bi olakšale posao. To je slučaj u ovome sustavu.

HTTPFlow je objekt koji predstavlja jednu HTTP transakciju i ujedno objekt s kojim se najviše radi prilikom rada s mitmproxyem. HTTPFlow sastoji se od kolekcije objekata koji opisuju jednu HTTP transakciju pa je tako moguće iz njega izvući informacije i o zahtjevu i o odgovoru.

Funkcija request poziva se prilikom presretanja zahtjeva. U konkretnom slučaju kada korisnik pokrene aplikaciju te pošalje zahtjev za početnom stranicom aplikacije pokrenut će se funkcija request koja u argumentu prima objekt tipa HTTPFlow koji sadrži sve podatke o poslanom zahtjevu. Pošto je poželjno da isti taj zahtjev proslijedimo nadograđenoj verziji aplikacije, potrebno je stvoriti novi objekt tipa HTTPFlow koji će biti identičan primljenom. Python API nudi mogućnost izravnog kopiranja objekta te se tako stvara kopirana verzija zahtjeva.

Na kopiranoj verziji zahtjeva potrebno je učiniti nekoliko izmjena. Prva od tih je promjena odredišne adrese zahtjeva. Pod pretpostavkom da smo pokrenuli docker spremniku na istom računalu dovoljno je promijeniti odredišna vrata da bi zahtjev stigao na željeno odredište. Korisnik specificira vrata na kojima se nalazi nadograđena verzija aplikacije.

Web aplikacije u praksi koriste kolačiće za upravljanje sjednicama i općenito za praćenje korisnika prilikom korištenja aplikacije. Kolačić je mali komadić podataka kojeg poslužitelj šalje korisniku. Korisnik ga zatim šalje natrag poslužitelju na zahtjev kako bi se utvrdilo primjerice dolaze li dva zahtjeva s istog web preglednika u svrhu ostavljanja korisnika prijavljenim. Zbog toga nije dovoljno ostaviti kolačić koji se nalazi u originalnom zahtjevu. Kolačići se dobivaju u zaglavlju odgovora pod poljem *Set-Cookie*. Tako je na svaku pojavu polja *Set-Cookie* u zaglavlju potrebno dodati dobiveni kolačić u sljedećem zahtjevu. Ako se ne mijenjaju kolačići doći će do greške u radu aplikacije te vrlo vjerojatno prijava u aplikaciju neće biti moguća. Također kada se korisnik odjavljuje potrebno je obrisati kolačiće. Ako se kolačići ne obrišu i korisnik se odjavi iz originalne aplikacije, unutar nadograđene aplikacije on će i dalje ostati prijavljen.

Prilikom ispunjava određene forme unutar aplikacije, primjerice forme za prijavu u sustav, uz ispunjene podatke većinom se šalju dodatne informacije kako bi se spriječili napadi na aplikaciju. Većinom se radi o obrani protiv CSRF (*Cross-site request forgery*) napada. Dodatne informacije većinom se šalju u obliku tokena koji u sebi sadrže skup nasumično generiranih znakova. Kako bi se omogućilo ručno stvaranje zahtjeva koji izvršavaju takve kritične operacije potrebno je imati na raspolaganju tokene nado-

građene verzije aplikacije. Tokene pronalazimo u odgovorima na zahtjeve tipa GET. Većinom se nalaze unutar forme u input tagovima. Funkcija koja obavlja pronalazak tokena je scrape_response_for_tokens.

U ispisu 4.3. prikazan je sadržaj funkcije za dohvaćanje tokena. Za rad s HTML sadržajem stranica korištena je BeautifulSoup knjižnica koja nudi jednostavan oblik prolaska kroz sadržaj HTML-a. Funkcija prima sadržaj HTTP odgovora te prima naziv tokena kojeg treba izvući iz sadržaja odgovora. Funkcija se poziva unutar response funkcije, odnosno na dolazak svakog pojedinog odgovora na GET zahtjev.

Kako bi se olakšao prolazak kroz cijeli sadržaj stranice, izvlače se imena svih tagova koji postoje u stranici. Zatim se pronalaze svi tagovi toga tipa i provjeravaju se njihovi atributi. Kada se pronađe željeni token on se dodaje u mapu po ključu vlastitog naziva te se kao vrijednost za taj ključ postavlja vrijednost tokena. Mogući su slučajevi da token bude prikazan na stranici bez da mu je odmah poznata vrijednost primjerice: <tag token_name=token_name></tag>. Tek nakon toga kasnije će se nalaziti prava vrijednost tokena. Zbog toga je potrebno provjeriti radi li se o tom slučaju. Također vrijednost tokena može se nalaziti u nazivu atributa i u vrijednosti atributa pa je potrebno proći kroz sve slučajeve i pojavljivanja tokena u sadržaju odgovora.

```
def scrape_response_for_tokens(response, tokens):
        soup = BeautifulSoup(response, "html.parser")
        tag_list = [tag.name for tag in soup.find_all()]
        tag_set = set(tag_list)
        token_dict = {}
        for tag in tag_set:
                all_tags = soup.find_all(tag)
                for t in all_tags:
                        for k,v in t.attrs.items():
                                for token in tokens:
                                        if token in v:
                                                if tag == "meta":
                                                        if v == t.attrs["content"]:
                                                                continue
                                                        token_dict[v] = t.attrs["content"]
                                                else:
                                                        token_dict[v] = t.attrs["value"]
                                        elif token in k:
                                                token_dict[k] = t.attrs[k]
                                        else:
                                                continue
```

```
        return token_dict, bool(token_dict)
```

Ispis 4.3: Funkcija za dohvaćanje tokena iz nadograđene verzije aplikacije

Uz pronalazak tokena potrebno je stvoriti novo tijelo zahtjeva u koje će se dodati ispravni token. Najjednostavniji slučaj je kada su zahtjevi u čistom tekstualnom obliku. Tada je dovoljno obrisati staru vrijednost tokena te postaviti novu vrijednost. Također ako se radi o JSON obliku zahtjeva zamjena vrijednosti tokena svodi se na promjenu jedne vrijednosti s obzirom na to da se JSON veoma lako prevodi u rječnik u pythonu. U slučaju da se za enkodiranje tijela zahtjeva koristi URL encoded form, što se još naziva *percend encoding* potrebno je učiniti određene preinake.

U slučaju URL encoded forme, tijelo zahtjeva dohvaćeno iz HTTPFlow objekta oblika je: ime_1=vrijednost_1& ime_2=vrijednost_2 itd. Dovoljno je razdvojiti tekst po znaku "&" kako bi se dohvatila potrebna imena i vrijednosti. Naziv *percent encoding* dolazi od posebnog načina kodiranja specijalnih znakova. Znakovi tipa: "[", "]", "(", ")" i slični prikazani su posebnim kodom. Pošto se u praksi navedeni znakovi često pojavljuju u tijelu zahtjeva potrebno je dekodirati sadržaj tijela zahtjeva prije rada s njim.

U ispisu 4.4 nalazi se prikaz rada s *percent encodingom*. U pythonu postoji knjižnica urllib koja nudi velik spektar mogućnosti prilikom rada s *percent encodingom*. Pomoću njenog dodatka parse lako se dekodiraju vrijednosti posebnih znakova.

Funkcija split_body rastavlja dobiveno tijelo zahtjeva i stvara rječnik s kojim se dalje može raditi po principu ključ=vrijednost. Svi ključevi predstavljaju imena u tijelu zahtjeva, a vrijednosti predstavljaju vrijednosti tih imena.

Prilikom postavljanja novostvorenog tijela zahtjeva potrebno je izvršiti kompresiju i dekompresiju dobivenog teksta zahtjeva jer HTTPFlow objekt prima tijelo zahtjeva u bitovnom obliku, a ne u tekstualnom.

```
body_dict = split_body(request_body)
for k1,v1 in body_dict.items():
        for k2,v2 in self.token_dict.items():
                k1_compared = k1
                if "%" in k1:
                        k1_compared = urllib.parse.unquote(k1)
                if k2 in k1_compared:
                                body_dict[k1] = urllib.parse.quote(v2)
body = build_body(body_dict)
body = gzip.compress(bytes(body, 'utf-8'))
flow_copy.request.content = gzip.decompress(body)
```

Ispis 4.4: Prikaz rada s percent encodingom

Još jedan od učestalih oblika enkodiranja tijela zahtjeva je multipart form. U ispisu 4.5 prikazan je izgled tijela multipart form zahtjeva. Multipart form se koristi u slučajevima kada korisnik želi postaviti datoteke na Internet (engl. *upload*). Izgled tijela je specifičan i ručno bi bilo poprilično teško izvući korisne informacije iz zahtjeva te iz izvučenih informacija stvoriti jedan takav novi zahtjev. U pythonu postoji knjižnica pod nazivom request_toolbelt koja u sebi sadrži paket multipart. Unutar paketa multipart postoje enkoder i dekoder za multipart form koji uvelike olakšavaju upravljanje tim tipom podataka.

```
----------905191404154484336597275426 6
Content-Disposition: form-data; name="text"

text default
----------905191404154484336597275426 6
Content-Disposition: form-data; name="file1"; filename="a.txt"
Content-Type: text/plain

Content of a.txt.

----------905191404154484336597275426 6
Content-Disposition: form-data; name="file2"; filename="a.html"
Content-Type: text/html

<!DOCTYPE html><title>Content of a.html.</title>

----------905191404154484336597275426 6--
```

Ispis 4.5: Primjer izgleda tijela multipart form zahtjeva

U ispisu 4.6 nalazi se primjer korištenja paketa multipart koji je implementiran unutar sustava. Isječak programa stvara rječnik iz multipart form teksta. Polja pod nazivom "name" postavlja kao ključeve. Vrijednosti polja pod nazivom "name" koja se nalaze u retcima ispod postavljaju se kao vrijednosti u rječniku. Treba obratiti pažnju na početni redak multipart form tijela u kojem se nalazi nasumični broj. Navedeni nasumični broj zove se boundary i nalazi se u zaglavlju zahtjeva. Potrebno je nasumični broj izvući iz zaglavlja zahtjeva te ga postaviti u novostvoreni zahtjev. Također ako se unutar multipart form tijela nalaze tokeni, potrebno je token originalne aplikacije zamijeniti s tokenom nadograđene aplikacije.

```
content_type_header = flow.request.headers['Content-Type']
response = ''
multipart_dict = {}
```

```
for part in decoder.MultipartDecoder(flow.request.content, content_type_header).parts:
        response += part.text + "\n"
        for k,v in part.headers.items():
                if k.decode("utf-8") == "Content-Type":
                        continue


                v = v.decode("utf-8")
                multipart_dict[v[v.find("name") + len("name") + 1:]] = part.text
for k,v in multipart_dict.items():
        for k1,v1 in self.token_dict.items():
                if k1 in k:
                        multipart_dict[k] = self.token_dict[k1]
content_type_header = flow.request.headers["Content-Type"]
boundary = content_type_header[content_type_header.find("boundary") + len("boundary") + 1:]
m = encode_multipart_formdata(multipart_dict, boundary)
body = gzip.compress(bytes(m, "utf-8"))
flow_copy.request.content = gzip.decompress(body)
```

Ispis 4.6: Prikaz rada s percent encodingom

Komponenta za manipulaciju zahtjevima prosljeđuje primljene odgovore komponenti za uspoređivanje odgovora. Odgovori se pohranjuju u rječniku. rječnik je struktutiran kao rječnik rječnika. Razlog takve strukture leži u činjenici da nije dovoljno stvoriti rječnik s ključem koji je primjerice put do resursa web aplikacije (engl. *path*). Moguć je slučaj stvaranja više zahtjeva za istom lokacijom za redom i zbog toga mogu nastati problemi u mapiranju ispravnih parova zahtjeva. U svrhu rješavanja tog problema komponenta za manipulaciju zahtjevima postavlja polje u zaglavlju zahtjeva pod nazivom *ordering number*. Tako će parovi istih zahtjeva imati jednake redne brojeve. Redni broj je dodatni ključ koji osigurava da komponenta za uspoređivanje odgovora uspoređuje ispravan par odgovora. Konkretan izgled rječnika mogao bi se prikazati na sljedeći način. Neka znakovi "{" i "}" predstavljaju granice jednog rječnika, a znakovi "[" i "]" predstavljaju granice liste. Onda se rječnik odgovora može prikazati kao:

{ path_1 : { ordering_number_1: [flow_11, flow_12] ,ordering_number_2: [flow_21, flow_22] } }.

Tako je omogućeno da se na istu lokaciju šalje više zahtjeva, neovisno o redoslijedu.

Kašnjenje u primanju parova odgovora može biti problem prilikom korištenja sustava. Dva para odgovora ne moraju doći istovremeno. U trenutku kada se komponenti za uspoređivanje odgovora prosljeđuju parovi odgovora moguće je da je jedan odgovor stigao, a drugi nije. Kada se dogodi takav slučaj zabilježe se potrebni ključevi za

kasniji pronalazak tog para te se prelazi na drugi par koji je na raspolaganju.

4.3. Uspoređivanje odgovora

Druga potkomponenta međukomponente je komponenta za uspoređivanje odgovora. Odgovori se ne uspoređuju u stvarnom vremenu već se svakih nekoliko sekundi pokreće komponenta za uspoređivanje odgovora koja uspoređuje dobivene odgovore. Broj sekundi nakon kojih se pokreće komponenta ovisi isključivo o korisniku koji ih zadaje. Komponenta za uspoređivanje odgovora pokreće se u zasebnoj dretvi što je u pythonu implementirano pomoću Threading knjižnice.

Komponenta za uspoređivanje odgovora posjeduje dva načina rada ovisno o tipu podataka koji se nalazi u zaglavlju Content-Type. Tekstualni oblici primjerice HTML i JSON imaju poseban način uspoređivanja.

Korisnik mora specificirati dozvoljene razlike kako bi sustav mogao ispravno uspoređivati parove odgovora te odrediti je li odgovor nadograđene aplikacije ispravan. Drugim riječima postoje li u odgovoru nadograđene aplikacije razlike koje nisu u skupu dozvoljenih razlika. Korisnik uči o dozvoljenim razlikama korištenjem sustava. Sustav prilikom svog rada generira datoteke u kojima se nalaze razlike parova odgovora. Lako se može procijeniti radi li se o razlici koja ovisi o slučajno generiranim brojevima ili ne. Većinom se pojavljuju ključne riječi kao što su: csrf_token, uuid, authentication_token, Request_Vericification_Token itd.

Uspoređivanje tekstualnih tipova podataka posebno je složeno za HTML. Ako aplikacija dohvaća JavaScript kod on u načelu mora biti u potpunosti identičan u oba slučaja. Uspoređivanje para HTML odgovora koji su ispravni će svejedno posjedovati razlike.

Za uspoređivanje tekstualnih tipova odgovora koristi se diff_match_patch knjižnica koja generira tekstualne razlike na poseban način. Primjer jednog ispisa razlika između dvaju odgovora nalazi se u ispisu 4.7. Retci koji počinju znakom "-" označuju dio teksta koji se nalazi u prvom odgovoru, a da se razlikuje od drugog odgovora. Retci koji počinju znakom "+" označuju dio teksta koji se nalazi u drugom odgovoru, a da se razlikuje od prvog odgovora.

Komponenta za uspoređivanje odgovora uzima dijelove teksta koji predstavljaju razlike. Prolazi kroz sve dijelove teksta, primjerice sve HTML tagove. Za Kada se pronađe pojavljivanje tog dijela teksta pretražuje se cijeli HTML tag kako bi se ustanovilo radi li se o dozvoljenoj razlici. U slučaju da se negdje u tagu pojavljuje ključna riječ koju je korisnik specificirao kao dozvoljenu razliku, sustav će označiti odgovor

kao ispravan. U protivnom odgovor je neispravan.

```
@@ -451,46 +451,46 @@
 n: %22
-f26662bd88be28b9ed06653c15056388a1b204
+1680878c140e6af3de428a09aa68ee3a2388c9
 79o%22
@@ -1308,48 +1308,48 @@
 ue=%22
-694f2f2907aa20a8b481ae0d1342344f5687fc4d
+315bfb9e3d266d0d868e062f23dc261c1906318b
 o158
```

Ispis 4.7: Primjer izgleda ispisa generiranog od strane diff_match_patch knjižnice

Komponenta za uspoređivanje odgovora koristi izlaz diff_match_patch alata kako bi pronašla mjesto u odgovoru gdje se pojavljuje sadržaj naveden pod oznakom "+" ili "-". Tako se dobiva ispis točnog mjesta u odgovoru na kojeg se korisnik mora fokusirati. Primjerice ispis prikazuje HTML tag unutar kojega se nalazi navedeni sadržaj te tako korisnik može zaključiti koja bi to razlika mogla biti. Primjer ispisa konkretnih položaja razlika u odgovoru nalazi se u ispisu 4.8. Na temelju ispisa korisnik može lako zaključiti da se razlike nalaze unutar atributa csrf_token te da se radi o dozvoljenim razlikama.

```
token: b0008cae11221c91b9a8f65d17c33202c819c5fa pronaen je u tagu:
<script type="text/javascript">
    var odoo = {
                        csrf_token: "b0008cae11221c91b9a8f65d17c33202c819c5fao",
                    };
  </script>

token: f1e1f925ded4142371379262d9b7948a23f2cc35 pronaen je u tagu:
<script type="text/javascript">
    var odoo = {
                        csrf_token: "f1e1f925ded4142371379262d9b7948a23f2cc35o",
                    };
  </script>

token: afcc10b704a9b72590bf16a0981f0930cc0670e pronaen je u tagu:
<input name="csrf_token" type="hidden" value="3afcc10b704a9b72590bf16a0981f0930cc0670eo1591702412"/>

token: 375e6ba85b3ad6abf80b6d618823feddf4cc339 pronaen je u tagu:
<input name="csrf_token" type="hidden" value="3375e6ba85b3ad6abf80b6d618823feddf4cc339o1591702412"/>
```

Ispis 4.8: Primjer ispisa razlika generiranog od strane sustava

Usporedba JSON odgovora ne koristi diff_match_patch knjižnicu. JSON objekte lako se prevodi u python rječnike. Zbog toga se JSON odgovori uspoređuju kao python riječnici uz određene posebne slučajeve. JSON objekti mogu biti složene strukture. Kada se JSON objekti prevode u rječnike mogući su slučajevi da se u jednom rječniku nalazi lista drugih rječnika. Zbog toga je usporedba rječnika implementirana rekurzivnim putem kako bi se pronašla konkretna razlika u riječnicima. Ako su riječnici navedene složenije strukture njihova usporedba neće dati razliku u najdubljoj razini rječnika. Primjerice imamo li rječnik u rječniku i neka je razlika u unutarnjem rječniku. Usporedba neće reći na kojem se točno ključu nalazi razlika u unutarnjem rječniku, nego će samo reći da se razlika nalazi na ključu čija je vrijednost unutarnji rječnik.

Funkcija iterate_dict prikazana na ispisu 4.8 implementira rekurzivni prolaz rječnikom te daje ispravne razlike. Funkcija prolazi po svim razinama rječnika dok god postoji razlika u vrijednostima. Unutar funkcije iterate_dict implementiran je jedan poseban slučaj. Kada unutar jednog rječnika postoji lista drugih rječnika, redoslijed rječnika u listi ne mora biti očuvan. Usporedba listi ovisna je o redoslijedu. Tako je moguće da postoje dvije liste označene kao različite, a zapravo se radi o identičnim riječnicima koji su poredani drugim redoslijedom. U svrhu spriječavanja navedenog problema liste se pretvaraju u tekstualni oblik i sortiraju se. Ako su sortirani tekstualni oblici liste identični onda su i njihove vrijednosti iste. Naravno da to nije nužno ispravan način usporedbe, ali u kontekstu ovoga rada vjerojatnost događanja slučaja u kojem je navedena usporedba pogrešna je minimalna. Funkcija na kraju daje ispis po uzoru na ispis diff_match_patch implementacije.

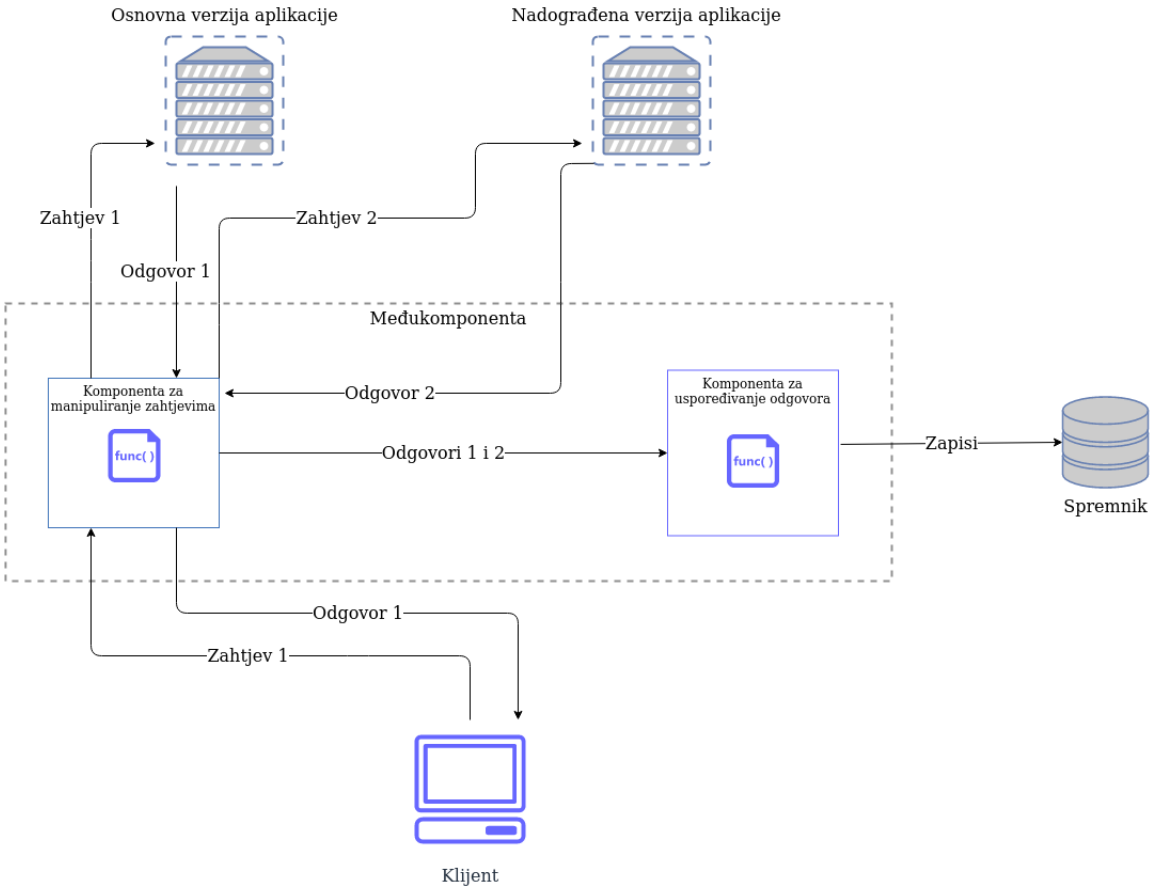
```
def iterate_dict(first_dict, second_dict, file, cnt):
    for k1,v1 in first_dict.items():
        for k2,v2 in second_dict.items():
            if k1 == k2:
                if v1 != v2:
                    if isinstance(v1, dict) and isinstance(v2, dict):
                        iterate_dict(v1,v2, file, cnt)
                    elif isinstance(v1, list) and (v2, list):
                        if len(v1) > 0 and len(v2) > 0:
                            if isinstance(v1[0], dict) and
                            isinstance(v2[0], dict):
                                for value1, value2 in zip(v1,v2):
                                    iterate_dict(value1, value2,
                                    file, cnt)
                    else:
```

```
                if isinstance(v1, str) and isinstance(v2, str):
                    first_sorted_string = str(''.join(sorted(v1)))
                    second_sorted_string = str(''.join(sorted(v2)))
                    if first_sorted_string == second_sorted_string:
                        continue
                file.write("-" + str(k1) + "=" + str(v1) + "\n+" + str(k2)
            + "=" + str(v2) + "\n")
```

Ispis 4.9: Prikaz rekurzivnog prolaska riječnicima

4.4. Sustav kao cjelina

Sustav kao cjelina može se prikazati kao na slici 4.4. Međukomponenta djeluje kao proxy sastavljen od dva dijela. Komponenta za manipulaciju zahtjevima presreće zahtjeve, stvara nove zahtjeve te je u načelu proxy skripta. Također komponenta za manipulaciju zahtjevima čita konfiguracijsku datoteku koju korisnik specificira prije pokretanja sustava.



Slika 4.4: Sustav kao cjelina

Konfiguracijska datoteka sastoji se od popisa dozvoljenih razlika te dijelova odgovora koji je potrebno izvući iz odgovora u svrhu uspješnog rada sustava. Radi se o tokenima koji se šalju unutar POST zahtjeva koje sustav izvlači iz odgovora pri svakom GET zahtjevu. Izgled konfiguracijske datoteke prikazan je u ispisu 4.9. Retci koji započinju sa znakom ":" označavaju dozvoljene razlike. Retci koji započinju sa znakom "+" označavaju dijelove odgovora koje sustav mora izvući kako bi mogao ispravno slati POST zahtjeve.

```
: csrf_token
: session_info
: write_date
: create_date
: last_update
:search_view
: date
: context
+csrf_token
```

Ispis 4.10: Izgled konfiguracijske datoteke

5. Eksperimenti

U ovom poglavlju prikazani su provedeni eksperimenti nad nekolicinom web aplikacija. Navedene su informacije naučene provođenjem eksperimenata. Svaka aplikacija ukratko je opisana i prikazani su osnovni ekrani ilustracije radi.

Izgradnja sustava tekla je iterativnim postupkom. Postupak se sastojao od pretraživanja web aplikacija, njihove instalacije kako bi se omogućila paralelizacija te testiranja sustava nad web aplikacijom. Poglavlje prikazuje četiri web aplikacije korištene prilikom izgradnje sustava:

- eShopOnWeb
- ColourAPI
- Odoo
- OpenProject

5.1. eShopOnWeb

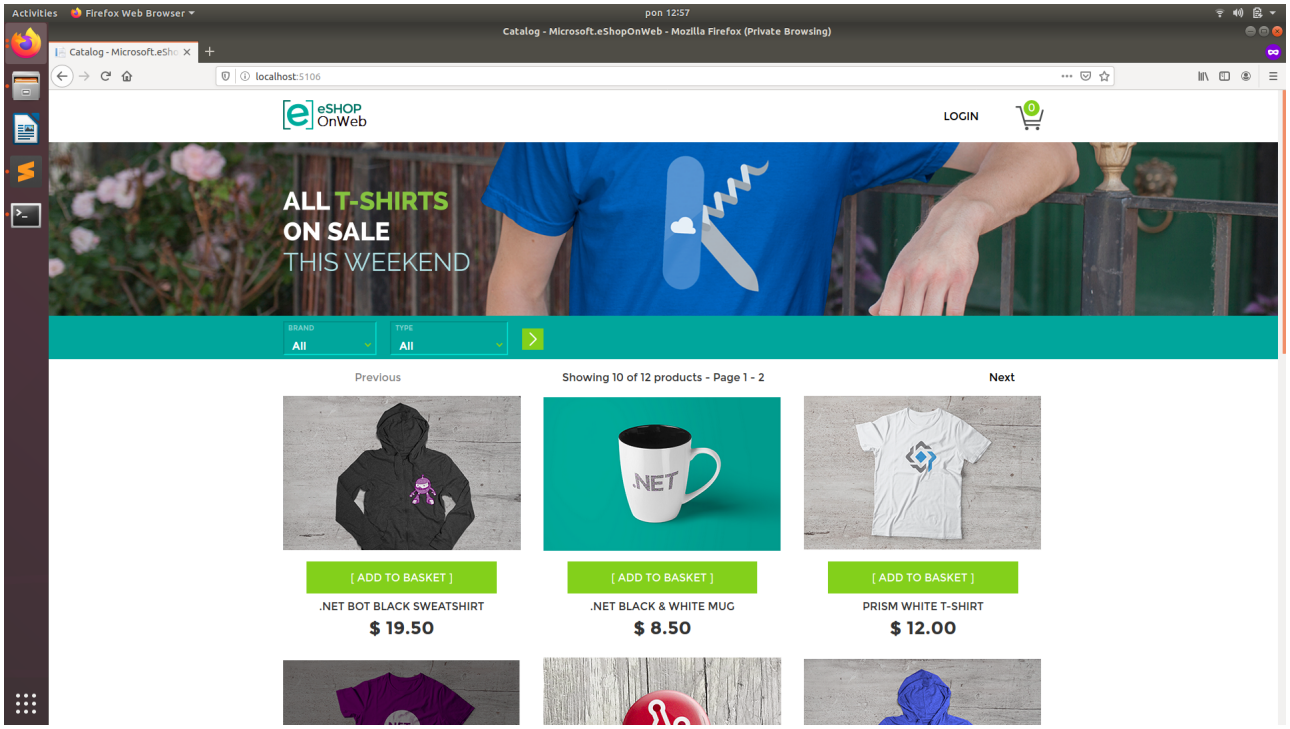
EShopOnWeb je primjer ASP.NET Core 3.1 referentne aplikacije koju pokreće Microsoft, demonstrirajući slojevitu arhitekturu aplikacija s monolitnim modelom implementacije [15].

Aplikacija je koncipirana kao primjer web trgovine. Osnovna namjena aplikacije je prikaz oblikovnih obrazaca u izgradnji ASP.NET Core web aplikacija. EShopOnWeb nije predviđen za stvarnu uporabu kao web trgovina već je idejno stvoren kao edukativni materijal. Zbog toga ne pruža širok spektar funkcionalnosti.

HTTP promet sastoji se od GET i POST zahtjeva čiji odgovori su pisani u HTML obliku. Odgovori sadrže HTML sadržaje stranica. Tijela zahtjeva kodirana su urlencodingom. Struktura prometa može se ocijeniti kao relativno jednostavna pošto nema JSON oblika niti multipart form zahtjeva.

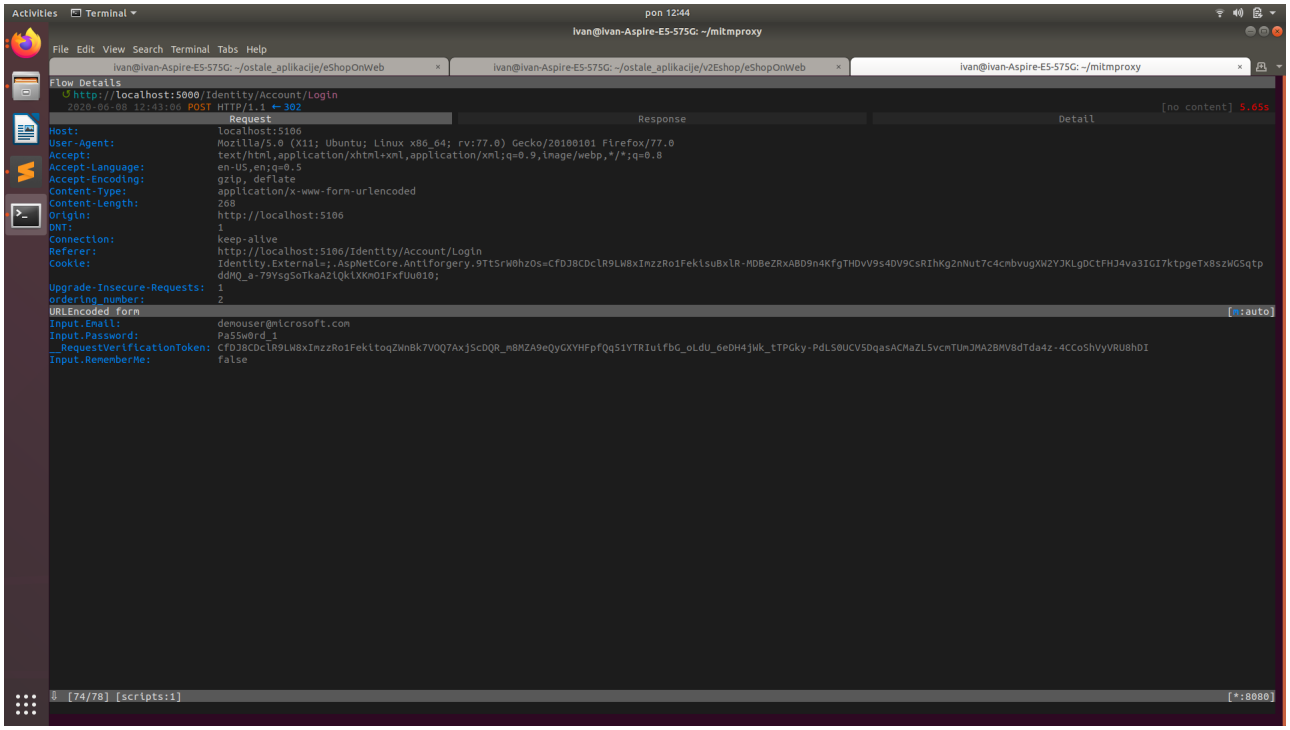
Početna stranica web aplikacije prikazana je na slici 5.1. Na početnoj stranici prikazani su artikli koje je moguće dodavati u košaru. Kako bi se izvršila akcija kupovine

potrebno se prijaviti u sustav. Prilikom usporedbe sadržaja početnih stranica dvaju verzija eShopOnWeb aplikacije jedine razlike nalaze se u polju pod naziv RequestVerificationToken. RequestVerificationToken je komadić nasumično generiranog teksta koji služi za borbu protiv CSRF napada. Kako bi omogućili ispravan rad s dvije verzije eShopOnWeb aplikacije potrebno je na svaku pojavu RequestVerificationTokena izvući njegovu vrijednost iz sadržaja stranice te ju pohraniti.



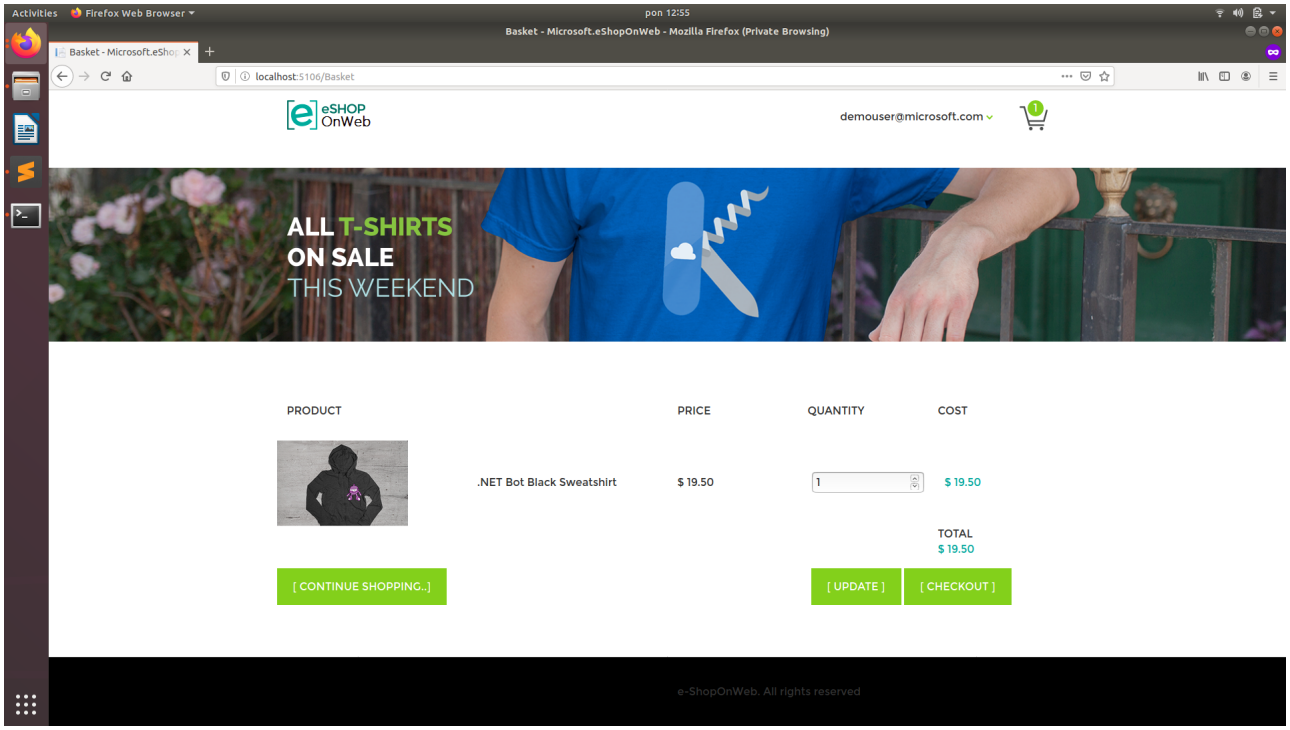
Slika 5.1: Početna stranica eShopOnWeb aplikacije

Prilikom slanja svakog POST zahtjeva u tijelu zahtjeva šalje se RequestVerificationToken. RequestVerificationToken je ujedno i dozvoljena razlika pošto ovisi o slučajnim brojevima. Na slici 5.2. prikazano je tijelo zahtjeva prilikom prijave u aplikaciju. Tijelo zahtjeva prijave sastoji se od korisničkih podataka korisnika koji se prijavljuje i RequestVerificiationTokena. Tijelo zahtjeva koji je upućen prema originalnoj aplikaciji prikazano je u obliku čistog teksta. Zbog toga je jednostavno izvući korisničke podatke. Uz prepisivanje korisničkih podataka dovoljno je zamijeniti vrijednost RequestVerificationTokena na vrijednost koja se izvlači iz sadržaja stranice nadograđene aplikacije. Nakon što sustav obavi navedene akcije moguće je obaviti prijavu u nadograđenu verziju aplikacije istovremeno dok se obavlja prijava u originalnu verziju aplikacije.



Slika 5.2: Tijelo zahtjeva prilikom prijave u eShopOnWeb

Nakon prijave u sustav moguće je kupovati željene artikle. Prilikom kupovine iz sadržaja stranice potrebno je izvući vrijednost ključa artikla. Vrijednost ključa prikazana je u obliku: Items[i].Key gdje i predstavlja indeks artikla u košarici. Zbog toga Items[i].Key spada u skup dozvoljenih razlika. Njegovu vrijednost treba izvlačiti iz sadržaja odgovora prilikom svake pojave jer je moguć slučaj da vrijednost ključa za isti artikl bude različita u različitim verzijama aplikacije. Na slici 5.3. prikazan je izgled košarice. Uz kupovinu artikala i prijavu u sustav, korisnik može mijenjati korisničke podatke i pregledavati izvršene transakcije.



Slika 5.3: Izgled košarice

Konfiguracijska datoteka eShopOnWeb aplikacije prikazana je u ispisu 5.1. Konfuguracijska datoteka sadrži RequestVerificationToken i Items oznake kao dozvoljene razlike te ih je potrebno izvlačiti iz sadržaja odgovora za ispravan rad sustava. Pošto je struktura HTTP prometa poprilično jednostavna i nema velikih razlika u sadržaju različitih odgovora ne pojavljuju se nikakvi drugi sadržaji koji ovise o generiranju slučajnih brojeva.

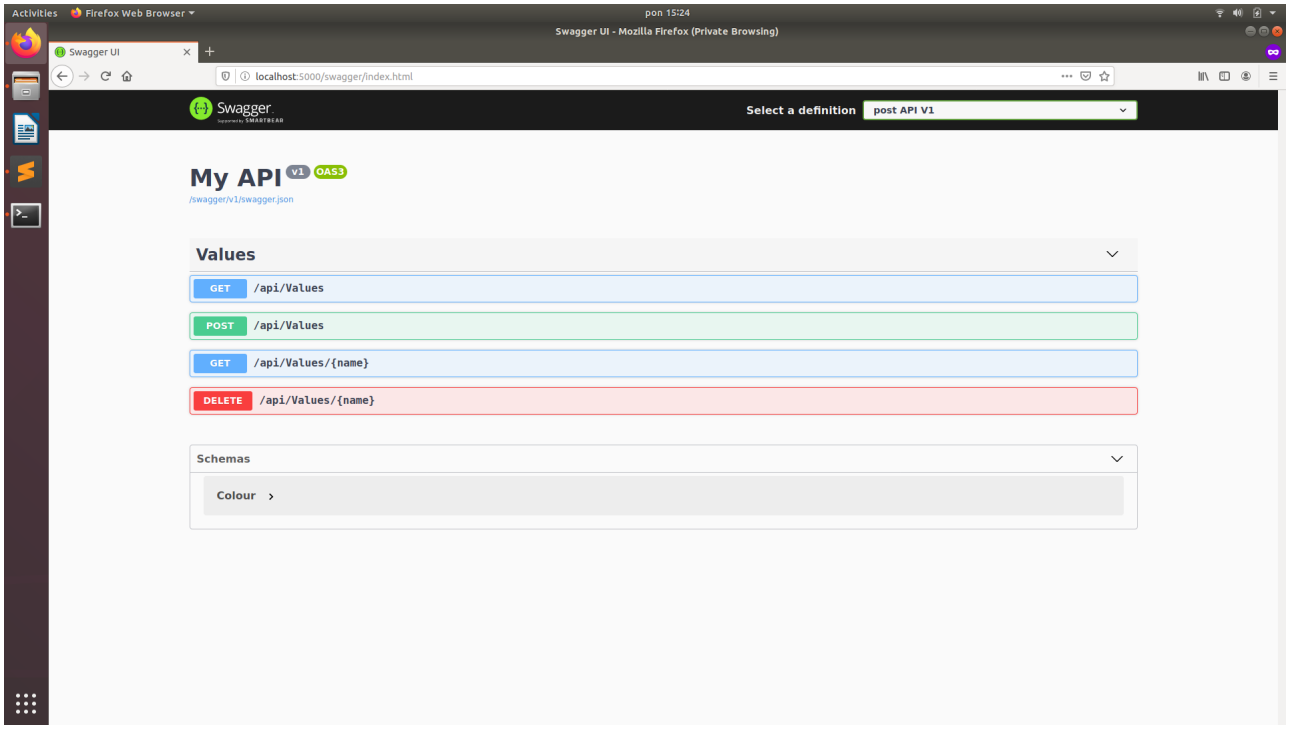
: RequestVerificationToken
:Items[
+ RequestVerificationToken
+Items[

Ispis 5.1: Izgled konfiguracijske datoteke eShopOnWeb aplikacije

5.2. ColourAPI

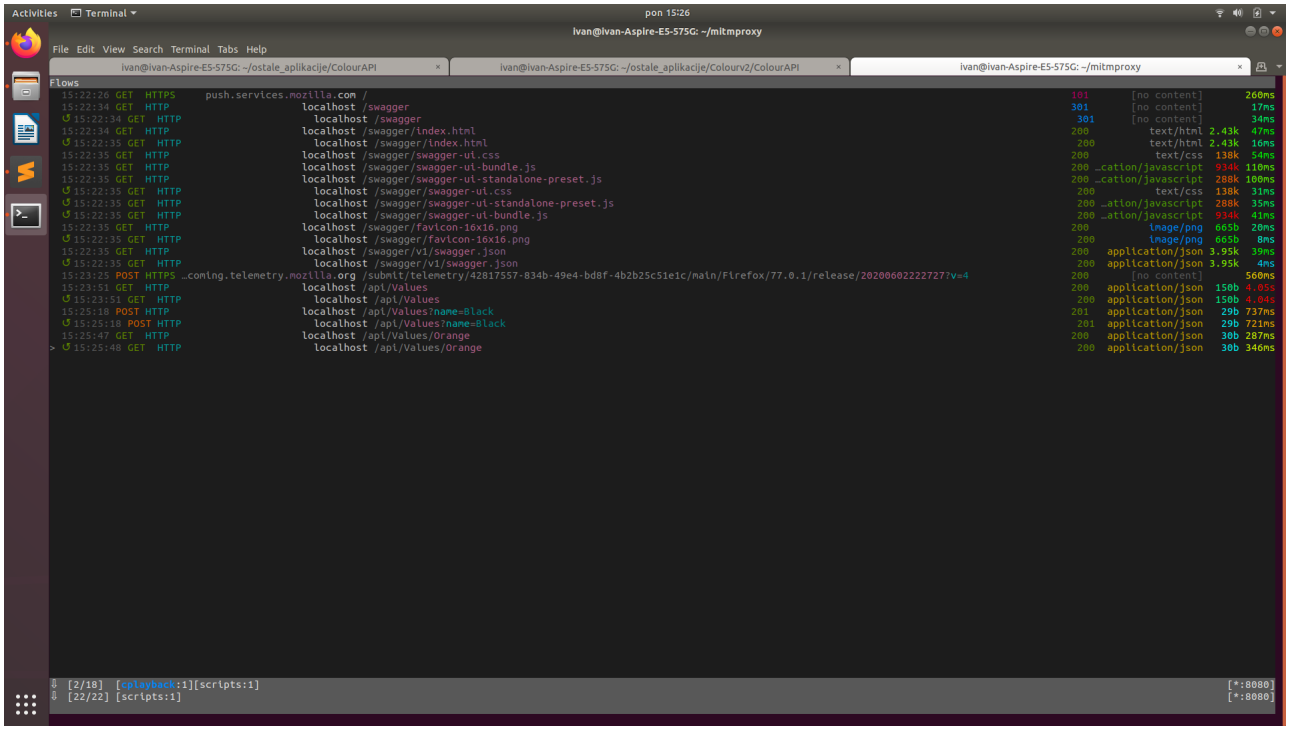
ColourAPI jednostavan je primjer REST API sučelja implementiranog u ASP .NET Coreu. Cilj testiranja bio je otkriti mogućnosti upotrebe sustava u radu s REST API sučeljem.

ColourAPI implementira CRUD operacije (engl. *Create Read Update Delete*) za jednu tablicu baze podataka pod nazivom Colour. Swagger je implementiran kao grafičko sučelje prema korisniku. Izgled početne stranice ColourAPI aplikacije prikazan je na slici 5.4. Pomoću Swaggera lako je izvršiti sve CRUD operacije nad aplikacijom.



Slika 5.4: Početna stranica ColourAPI aplikacije

Strutura HTTP prometa svodi se na slanje upita (engl. *query*) i primanja JSON odgovora unutar kojeg su zapakirani objekti. Objekti su boje koji se sastoje od jedinstvenog identifikatora i naziva. HTTP promet prikazan je na slici 5.5.



Slika 5.5: Struktura HTTP prometa ColourAPI aplikacije

Pošto se JSON pokazao kao idealan za uspoređivanje na način opisan u poglavlju Implementacija, rad s REST API sučeljem poprilično je jednostavan. U skup dozvoljenih razlika dovoljno je uvrstiti identifikator objekta koji predstavlja boju, odnosno id. Identifikator ne mora nužno biti različiti. U početnom stanju aplikacije identifikatori će biti jednaki. Daljnjim korištenjem aplikacije te dodavanjem i brisanjem boja moguće je da će baze podataka poprimiti različita stanja i dodijeliti različite identifikatore za istu boju. U radu aplikacije ne koriste se tokeni stoga će se unutar konfiguracijske datoteke nalaziti samo id i to u skupu dozvoljenih razlika, ali ne u skupu sadržaja kojeg treba izvući iz stranice. Pošto se sva dohvaćanja primjera iz baze vrše pomoću upita koji sadrže naziv boje nije potrebno izvlačiti identifikatore.

REST API pokazao se kao dobro područje primjene za izrađeni sustav. Jednostavnost strukture prometa i svojstvo JSON-a da se lako prevodi u python rječnik uvelike idu u prilog tome.

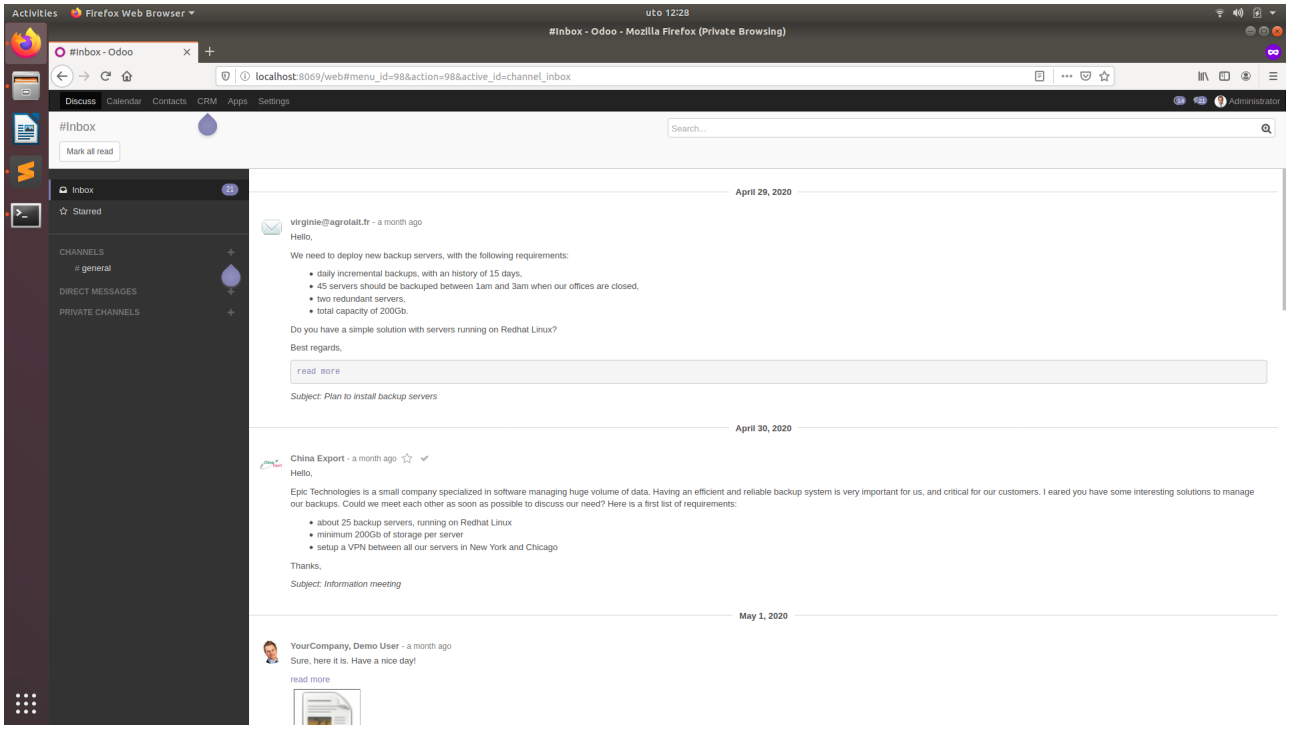
5.3. Odoo CRM

Odoo je potpuno integrirani i prilagodljivi sustav otvorenog koda. Aplikacija uključuje prodaju, CRM, upravljanje projektima, proizvodnju, zalihe, računovodstvo i druge poslovne potrebe. Korištenjem Odoo aplikacije moguće je sve poslovne funkcije držati

na jednom mjestu. Tako se timovima omogućuje suradnja s drugim odjelima putem jedne ujedinjene platforme [16].

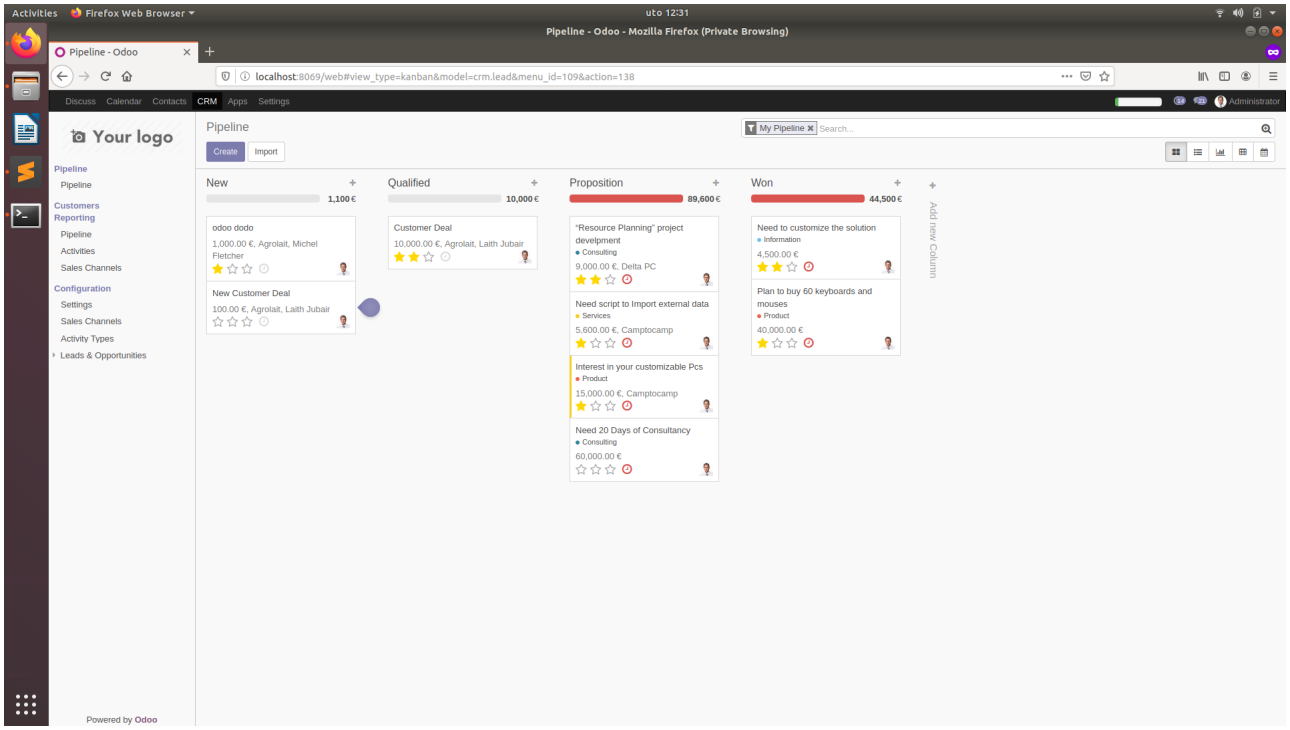
Za provođenje eksperimenata korišten je Odoo 11.0 CRM. Odoo CRM je modul za upravljanje odnosima s kupcima unutar platforme Odoo. Odoo CRM omogućuje praćenje svih potencijalnih prilika na jednoj lokaciji.

Korisnik se nakon prijave sustav preusmjerava na početnu stranicu koja je prikazana na slici 5.6. Početna stranica koncipirana je kao popis mailova generiranih prilikom instalacije CRM sustava. U početnom stanju aplikacije korisnik može slati mailove pretpostavljenim klijentima generiranim pri instalaciji.



Slika 5.6: Početna stranica Odoo aplikacije

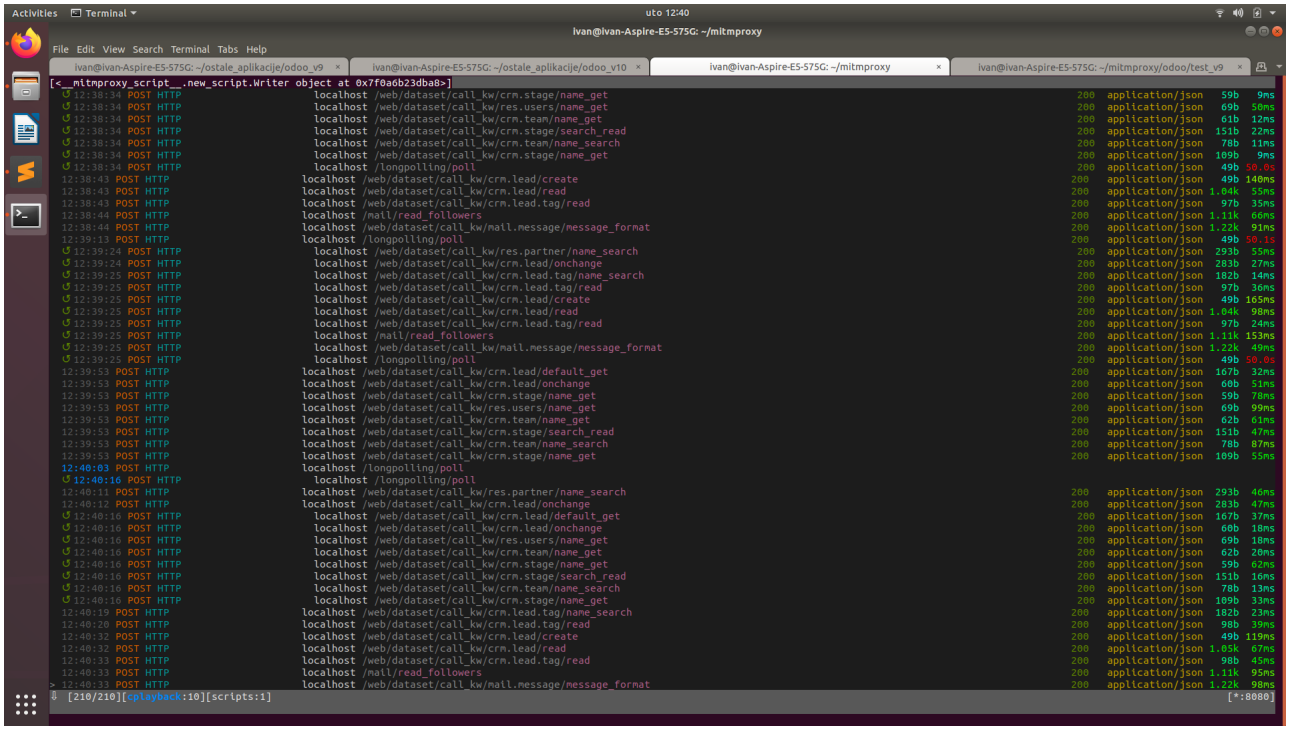
Otvaranjem kartice CRM prikazuje se preglednik prilika (engl. *Opportunities*). Unutar CRM kartice korisnik upravlja prilikama, stvara nove, uređuje postojće i briše nepotrebne. Općenito unutar CRM kartice korisnik upravlja odnosima s kupcima i izvršava sve željene akcije vezane uz CRM sustav. Izgled CRM sustava prikazan je na slici 5.7. Uz osnovne funkcije koje se vežu za CRM sustave, moguće je uređivati korisničke podatke, stvarati nove kontakte, slati poruke i slično.



Slika 5.7: CRM sustav

Struktura prometa Odoo aplikacije svodi se na izmjenu JSON objekata koji sadrže potrebne podatke za izvršavanje zadane akcije. Odgovori HTML oblika se rijetko pojavljuju. Jedina pojavljivanja HTML odgovora su prilikom učitavanja forme za prijavu u sustav i učitavanja početne stranice. Struktura prometa prikazana je na slici 5.8. Jedan od primjera izmjene JSON prometa je sljedeći. Primjerice, prilikom stvaranja nove prilike korisnik ispunjava podatke u formi za stvaranje prilike. Nakon pritiska na završavanje stvaranja prilike šalje se nekolicina zahtjeva koji u tijelu zahtjeva sadrže JSON objekte o stvorenoj prilici. Odgovori na navedene zahtjeve također sadrže JSON objekte. Za sve akcije vezane uz CRM izmjenjuju se JSON objekti na navedeni način.

Zbog velike zastupljenosti JSON prometa Odoo CRM bio je glavni izvor informacija o JSON prometu. Iz eksperimentiranja s Odoo CRM sustava izvučena je velika većina znanja o uspoređivanju JSON prometa te su svi navedeni slučajevi JSON usporedbe u poglavlju Implementacija izvučeni iz Odoo CRM aplikacije.



Slika 5.8: Struktura prometa Odoo aplikacije

Konfiguracijska datoteka uglavnom sadrži atribute koji se pojavljuju unutar JSON prometa zbog njegove velike zastupljenosti. Jedini atributi koji se pojavljuju unutar HTML prometa su: csrf_token i session_info. Csrf_token služi za obranu od CSRF napada. Csrf_token je potrebno izvlačiti iz sadržaja stranice zato što se šalje unutar tijela zahtjeva prilikom prijave i sličnih akcija. Session_info sadrži podatke o korisničkoj sjednici. Unutar session_info atributa nalazi se sjednički identifikator. Sjednički identifikator ovisi o slučajnim brojevima stoga session_info spada u skup dozvoljenih razlika.

Ispis 5.2 prikazuje isječak izlaza razlika u odgovorima na zahtjev za početnom stranicom Odoo aplikacije. Iz ispisa 5.2 lako se zaključuje kako su csrf_token i session_info razlike koje ovise o slučajnim brojevima te kako su specifične za svaku verziju aplikacije pa su zbog toga i dozvoljene u smislu razlika.

```
token: f1e1f925ded4142371379262d9b7948a23f2cc35 pronaen je u tagu:
<script type="text/javascript">
    var odoo = {
                        csrf_token: "f1e1f925ded4142371379262d9b7948a23f2cc35o",
                    };
  </script>

token: b0008cae11221c91b9a8f65d17c33202c819c5fa pronaen je u tagu:
```

```
<script type="text/javascript">
    var odoo = {
                                csrf_token: "b0008cae11221c91b9a8f65d17c33202c819c5fao",
                        };
  </script>

token: web.base.url": "http://localhost:8069", "server_version": "11.0−20200417", "company_id": 1, "username":
<script type="text/javascript">
    odoo.session_info = {"web.base.url": "http://localhost:8069", "server_version": "11.0−20200417", "company_
  </script>

token: partner_id": 3, "server_version": "11.0−20200417", "name": "Administrator", "currencies": {"1": {"positi
<script type="text/javascript">
    odoo.session_info = {"partner_id": 3, "server_version": "11.0−20200417", "name": "Administrator", "currencie
  </script>
```

Ispis 5.2: Isječak razlika u odgovorima Odoo aplikacije

Cijela konfiguracijska datoteka prikazana je u ispisu 5.3. Ostale razlike pripadaju JSON prometu. Atributi kao što su write_date, create_date, last_update i date su dozvoljeni u smislu razlika zbog kašnjenja koje se javlja prilikom istovremenog rada dvaju verzija aplikacija. U praksi originalna verzija aplikacije odgovara znatno brže od nadograđene pa se zbog toga pojavljuju razlike u vremenskim oznakama primjerice prilikom stvaranju novih prilika. Id predstavlja identifikator objekata povučenih iz baze. Pošto svaka verzija aplikacije radi s vlastitom bazom pojavljivat će se razlike u id vrijednostima, stoga se id također smatra dozvoljenom razlikom.

```
: csrf_token
: session_info
: write_date
: create_date
: id
: last_update
: search_view
: date
+csrf_token
```

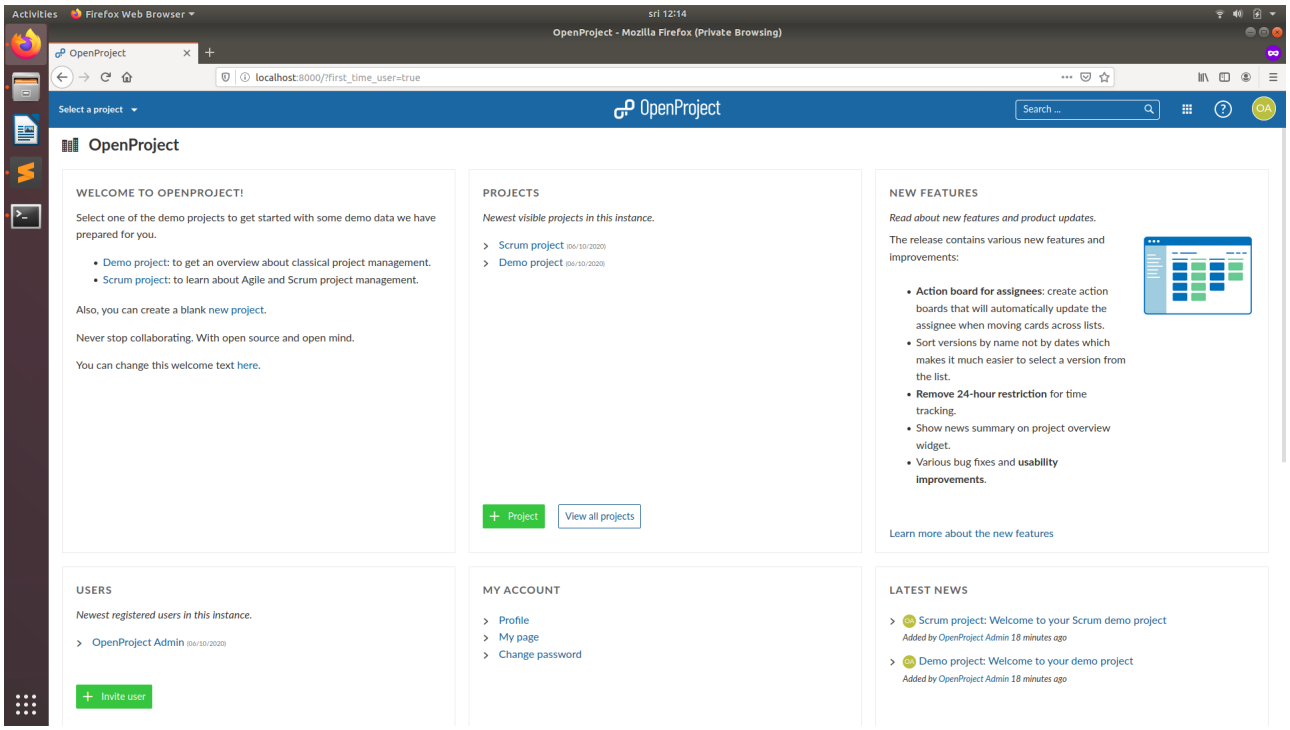
Ispis 5.3: Konfiguracijska datoteka Odoo aplikacije

5.4. OpenProject

OpenProject je internetski softver za upravljanje projektima. Njegove ključne karakteristike su: planiranje i organiziranje projekata, nacrt plana i planiranje izdavanja proizvoda, upravljanje zadacima i suradnja tima, Agile i Scrum, praćenje vremena, izvještavanje o troškovima, sastavljanje proračuna, praćenje grešaka, forum, dnevni red sastanka i minutaže sastanka [17].

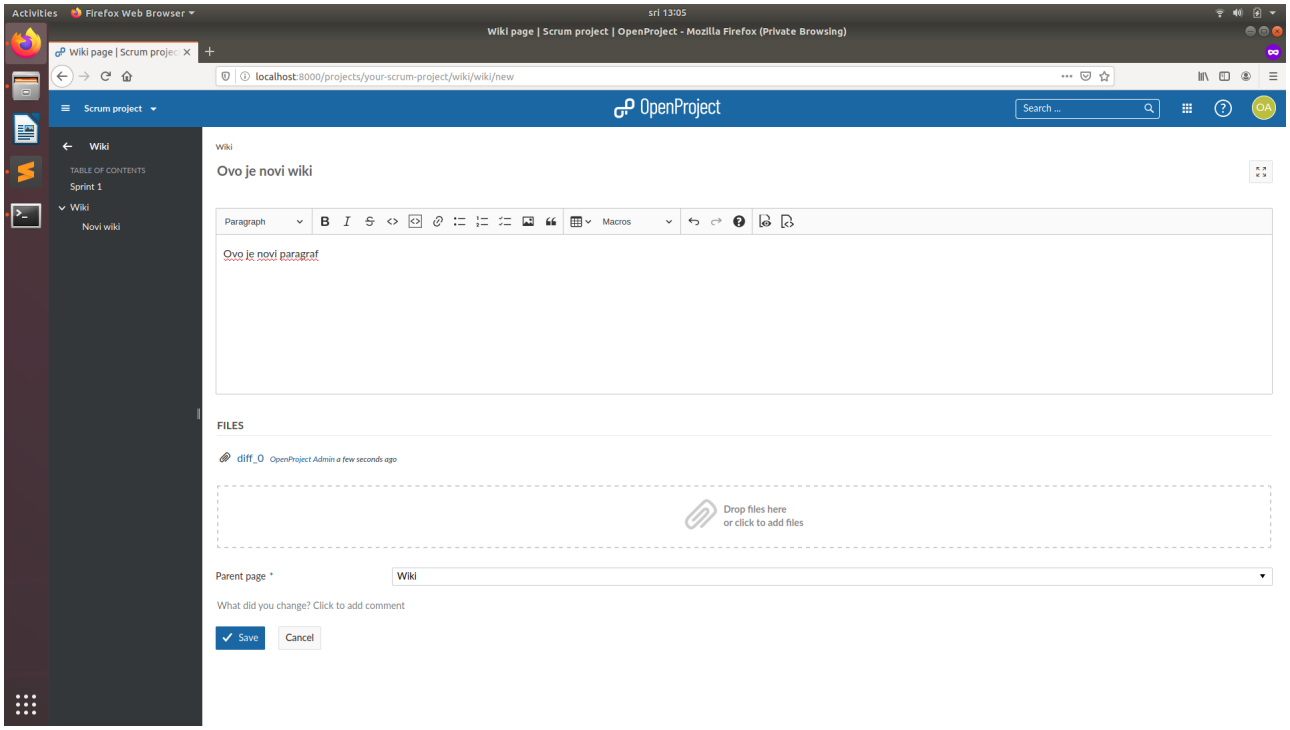
Dostupna je docker slika koja povezuje cijelu mrežu docker spremnika, stoga nije potrebno koristiti docker-compose alat. Dovoljno je pokrenuti docker sliku koja pokreće sve potrebne spremnike istovremeno.

Početna stranica prikazana je na slici 5.9. Nakon prijave u sustav korisnik se preusmjerava na prikazanu početnu stranicu. S početne stranice korisnik odabire projekte koje želi pregledavati.



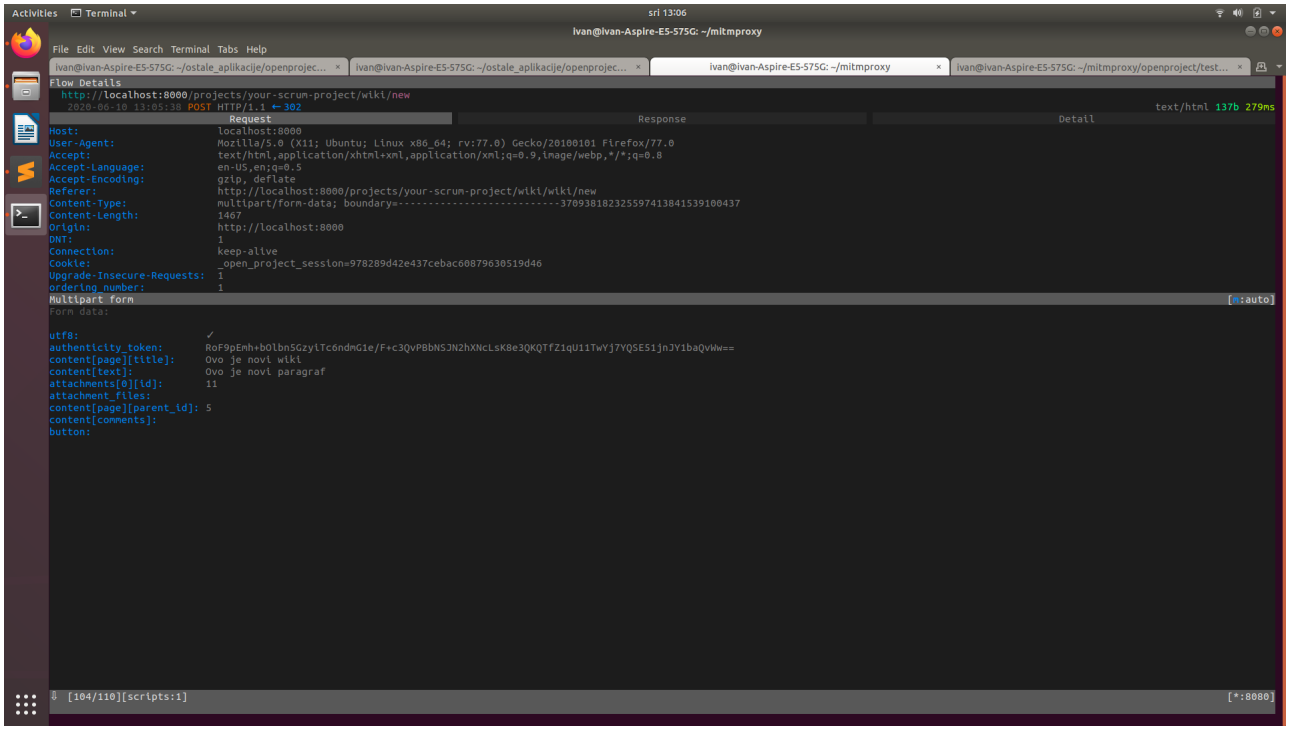
Slika 5.9: Početna stranica sustava OpenProject

Slanje POST zahtjeva razlikuje od dosadašnjih eksperimenata po tipu podataka koji se šalju u tijelu zahtjeva. OpenProject koristi multipart form kao tip tijela zahtjeva kako bi omogućio postavljanje datoteka na sustav. Primjerice, želi li korisnik stvoriti novi Wiki otvara se forma za stvaranje novog Wikija. Forma je prikazana na slici 5.10.



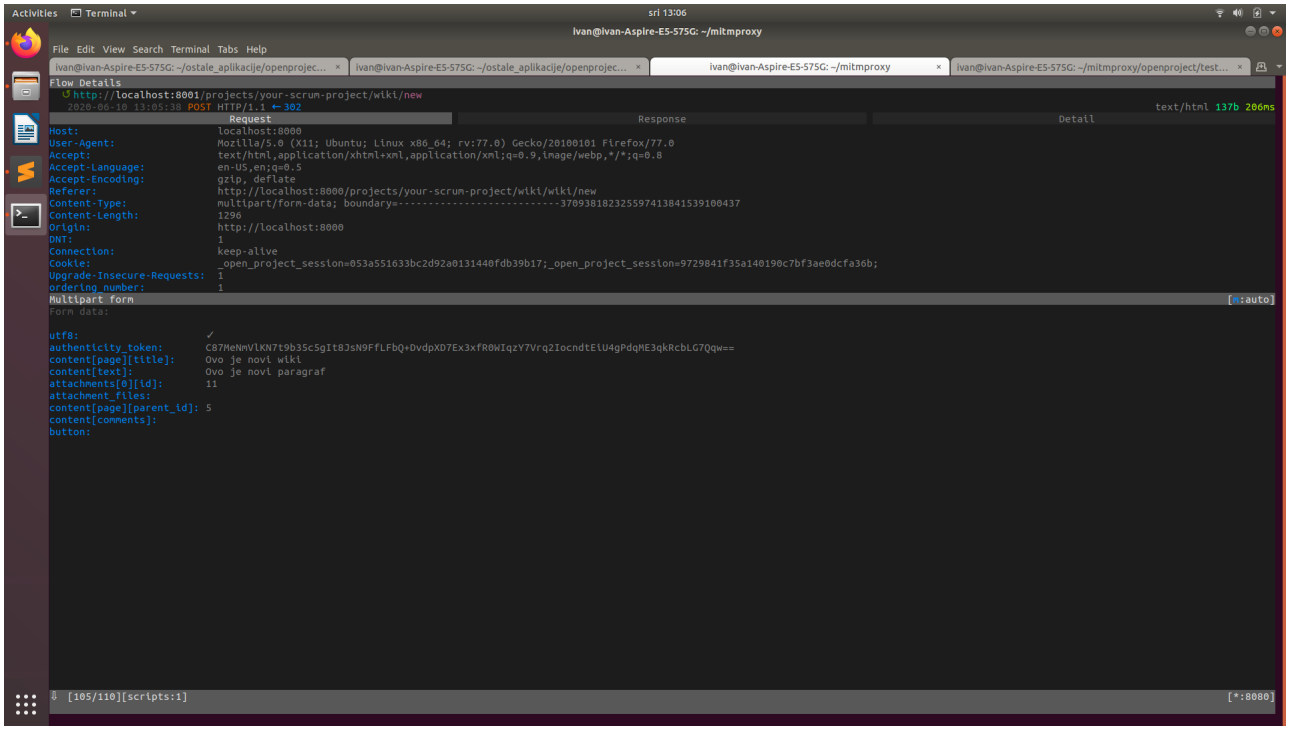
Slika 5.10: Stvaranje novog Wikija

Korisnik je postavio datoteku diff_0 i ispunio potrebna polja za stvaranje wikija. Kako bi se navedena akcija uspješno izvela na nadograđenoj verziji aplikacije koja je paralelno pokrenuta potrebno je omogućiti rad s multipart form tipom podataka. originalni zahtjev prikazan je na slici 5.11. Unutar tijela zahtjeva nalazi se authenticity_token koji je u principu identičan csrf_tokenu. Uz token nalazi se tekst unesen prilikom ispunjavanja forme te priložena datoteka koja nije vidljiva u ispisu pošto se radi o binarnom tekstu koji se ne prikazuje u izlazu sučelja mitmproxy.



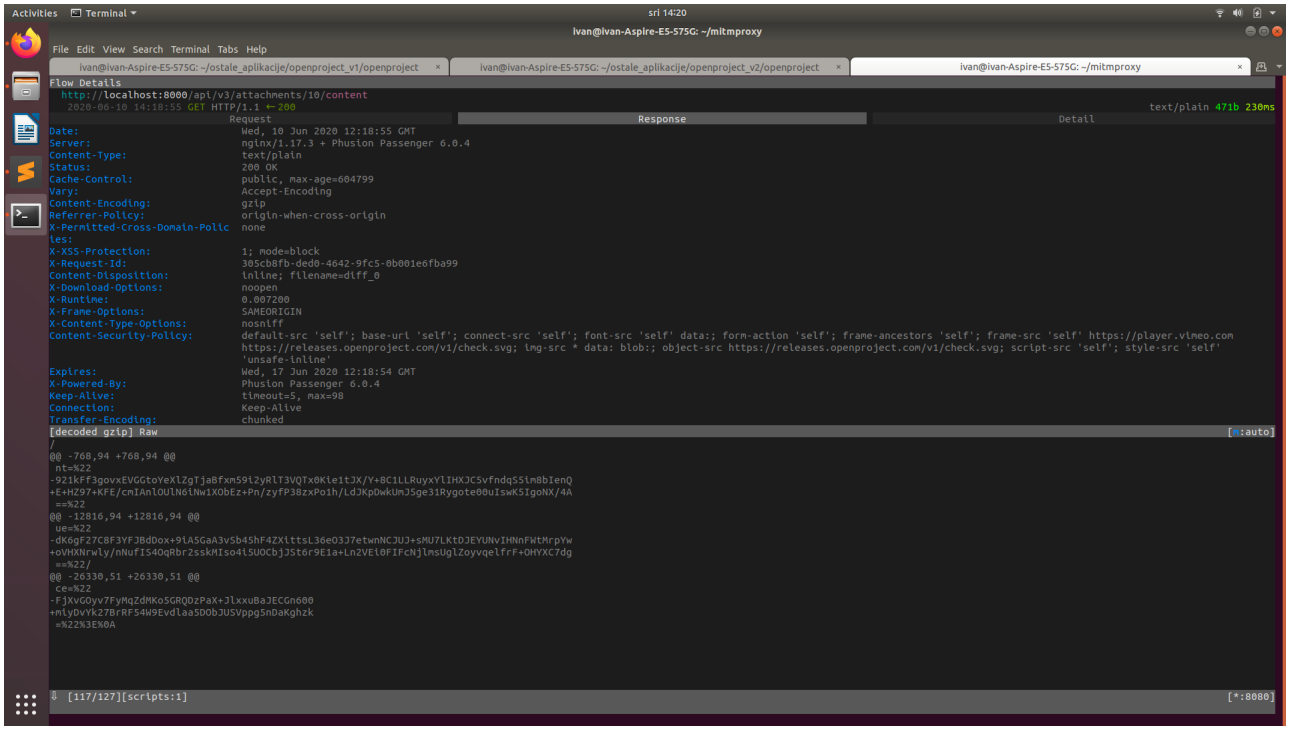
Slika 5.11: Zahtjev za stvaranje novog Wikija upućen originalnoj verziji aplikacije

Prilikom stvaranja novog zahtjeva koji će se uputiti nadograđenoj aplikaciji, potrebno je promijeniti vrijednost tokena, a sve ostale vrijednosti ostaviti nepromijenjene. Rad s multipart form podacima opisan je u poglavlju Implementacija. Novostvoreni zahtjev prikazan je na slici 5.12. Iz slike vidljivo je da je izmijenjen samo token, dok su ostale vrijednosti ostale jednake.

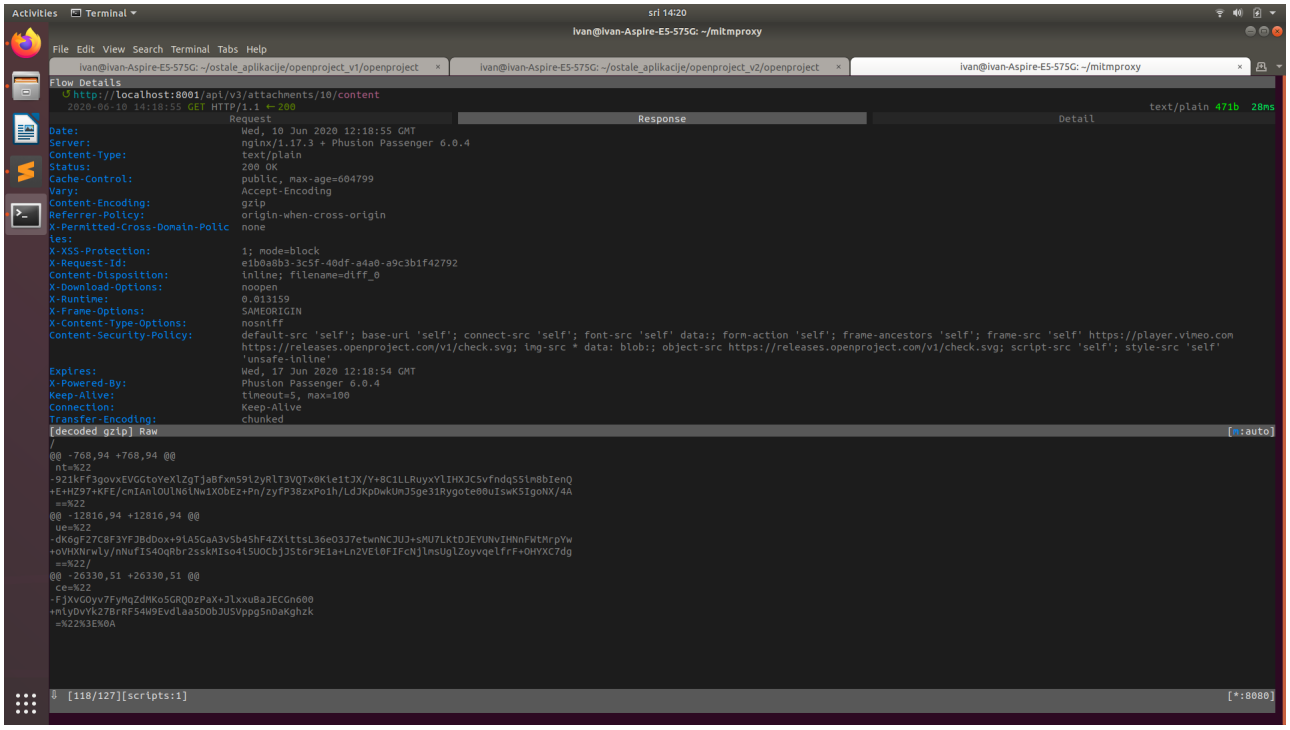


Slika 5.12: Novostvoreni zahtjev za stvaranje novog Wikija upućen nadograđenoj verziji aplikacije

Pošto vrijednost datoteke postavljene u privitku nije prikazana potrebno se uvjeriti da je ona stvarno postavljena na obe verzije aplikacije. Prilikom stvaranja novog wikija izmjenjuje se nekolicina zahtjeva i odgovora. Zahtjevi s ključnom riječi attachment prikazuju sadržaj postavljenog privitka prilikom stvaranja novog wikija. Na slikama 5.13 i 5.14 prikazani su odgovori na sadržaj privitka koji potvrđuju da je datoteka stvarno postavljena u obe verzije. Datoteka postavljena kao privitak je izlaz komponente za uspoređivanje odgovora koji prikazuje dijelove odgovora koji se razlikuju. Iz slika 5.13 i 5.14 vidi se da je datoteka uspješno postavljena te da je u oba slučaja postavljena uspješno i bez razlika u sadržaju datoteke.



Slika 5.13: Sadržaj privitka poslanog na originalnu aplikaciju



Slika 5.14: Sadržaj privitka poslanog na nadograđenu aplikaciju

Struktura prometa je ujednačena u smislu tipova podataka. Za slične zahtjeve izmjenjuje se nekolicina HTML i JSON odgovora. Za neke akcije izmjenjuju se zahtjevi

tipa multipart form. HTML zahtjevi razlikuju se u csrf_tokenu, nonceu, idu, keyu i određenim styles parametrima. Nonce je nasumično generiran broj koji se koristi samo jednom. Česta upotreba noncea je u autentifikacijskim protoklima i kriptografiji. Razlike u idu i keyu proizlaze iz korištenja različitih baza podataka. Styles parametar koristi se prilikom dohvaćanja potrebnih parametara za izgled stranice. Radi se o zahtjevima za css datotekama koje u nazivu sadrže nasumično generirani niz znakova. Obe verzije aplikacije unutar sadržaja HTML odgovora imaju različite vrijednosti nasumično generiranog niza znakova uz pojavu ključne riječi styles stoga se ona deklarira kao dozvoljena razlika. Cijela konfiguracijska datoteka prikazana je u ispisu 5.4. Ostale razlike vezane su uz JSON promet i vežu se uz razlike u vremenskim oznakama nastalim radi kašnjenja odgovora nadograđene aplikacije.

```
:token
:nonce
:uuid
:key
: styles
:timestamp
:createdAt
:updatedAt
+token
+nonce
```

Ispis 5.4: Konfiguracijska datoteka OpenProject aplikacije

5.5. Pouzdanost ispitivanja

Kako bi se ocijenila pouzdanost ispitivanja aplikacije uvodi se mjera pouzdanosti ispitivanja. Jednostavna pouzdanost ispitivanja svodi se omjer lokacija s kojima je vršena interakcija i svih lokacija u aplikaciji. Tako se dobiva postotak prolaznosti svim putevima u aplikaciji i kao takva mjera može se smatrati dovoljnom za mjeru pouzdanosti. Mjera pouzdanosti prikazana je u (5.1).

$$pouzdanost = \frac{ukupan\ broj\ testiranih\ puteva}{ukupan\ broj\ puteva\ u\ aplikaciji} \tag{5.1}$$

6. Zaključak

Izgradnja sustava za nadgledanu nadogradnju programskih komponenti složen je i zahtjevan postupak. Proces izgradnje je tekao iterativno te su se iz eksperimentiranja s novim web aplikacijama učile nove stvari koje su iskorištene prilikom implementacije. Izazovnost rješavanja ovoga problema je manjak sličnih primjera koji su na raspolaganju te je zbog toga cijela izgradnja sustava kretala od nule. U radu su prikazani i objašnjeni detalji implementacije sustava te ideje koje su vezane uz implementaciju. Također je navedena nekolicina primjera korištenja sustava.

Rezultati sustava na provedenim aplikacijama su zadovoljavajući. Zbog iterativnosti postupka svaka aplikacija donijela je nove probleme koji su se trebali riješiti. Sustav nije u potpunosti općenit pošto bi trebalo provesti velik niz eksperimenata na različitim tipovima aplikacija. Tako bi se izvuklo mnoštvo novih informacija i problema s kojima bi se trebalo suočiti. U daljnjem radu mogao bi se predložiti precizniji model računanja pouzdanosti ispravnog rada nadograđene verzije aplikacije. Također u svrhu podizanja kvalitete sustava moguće je dodavanja funkcionalnosti promjene verzije aplikacije u trenutku korištenja. Primjerice da korisnik po želji mijenja verziju aplikacije s kojom direktno komunicira putem web preglednika. Također moguće je ukomponirati rad sustava s razvojnim okolinama koje automatiziraju akcije korisnika. Tako bi se izbjegao vremenski ulog koji je potreban da bi se testirala željena aplikacija.

Ispravan rad sustava za sada je sveden na navedene aplikacije koje su objašnjene u poglavlju Eksperimenti. Također rad sustava je ograničen na tipove podataka koji su objašnjeni u poglavlju Implementacija. Nadogradnja sustava je moguća provođenjem novih eksperimenata po uzoru na do sad odrađene.

LITERATURA

[1] Dadzie, J., "Understanding Software Patching", 24.3.2005., [Na Internetu], Dostupno na: https://dl.acm.org/doi/pdf/10.1145/1053331.1053343, [Pristupljeno 1.6.2020.]
[2] "Patch (computing)", Wikipedia, 19.5.2020., Dostupno na: https://en.wikipedia.org/wiki/Patch_(computing), [Pristupljeno 1.6.2020.]
[3] Faus, N.L., Huff, P.D., Systems and Methods for Managing Software Patches, 16.4.2013., [Na Internetu], Dostupno na: https://patentimages.storage.googleapis.com/9a/11/73/e453c3574e3113/US8423993.pdf, [Pristupljeno 2.6.2020.]
[4] Koyuncu, A., Bissyandé T.F., Kim, D., Klein, J., Monperrus, M., Le Traon Y., "Impact of Tool Support in Patch Construction", 18.8.2017., [Na Internetu], Dostupno na: https://arxiv.org/pdf/1812.07416.pdf, [Pristupljeno 2.6.2020.]
[5] McKeeman, W.M., "Differential Testing for Software", 1998., [Na Internetu], Dostupno na: https://www.cs.swarthmore.edu/~bylvisa1/cs97/f13/Papers/DifferentialTestingForSoftware.pdf, [Pristupljeno 2.6.2020.]
[6] "Differential testing", Wikipedia, 20.4.2019., Dostupno na: https://en.wikipedia.org/wiki/Differential_testing, [Pristupljeno 2.6.2020.]
[7] Petsios T., Tang A., Stolfo S., Keromytis A.D, Jana S., "NEZHA: Efficient Domain-Independent Differential Testing", 2017., [Na Internetu], Dostupno na: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7958601, [Pristupljeno 2.6.2020.]
[8] Evans R.B., Savoia A., "Differential Testing : A new approach to change detection", 9.2007., [Na Internetu], Dostupno na: https://dl.acm.org/doi/pdf/10.1145/1295014.1295038, [Pristupljeno 2.6.2020.]
[9] "What is Docker", [Na Internetu], Dostupno na: https://opensource.com/resources/what-docker, [Pristupljeno 5.6.2020.]
[10] "What is a Container?", [Na Internetu], Dostupno na: https://www.docker.com/resources/what-container, [Pristupljeno 5.6.2020.]

[11] Boettiger C., "An introduction to Docker for reproducible research", 1.2015., [Na Internetu], Dostupno na: https://dl.acm.org/doi/pdf/10.1145/2723872.2723882, [Pristupljeno 5.6.2020.]
[12] "Build and run your image", [Na Internetu], Dostupno na: https://docs.docker.com/get-started/part2/, [Pristupljeno 5.6.2020.]
[13] Petters, J., "What is a Proxy Server and How Does it Work?", [Na Internetu], Dostupno na: https://www.varonis.com/blog/what-is-a-proxy-server/, [Pristupljeno 5.6.2020.]
[14] "mitmproxy", [Na Internetu], Dostupno na: https://mitmproxy.org/, [Pristupljeno 5.6.2020.]
[15] "Microsoft eShopOnWeb ASP.NET Core Reference Application", [Na Internetu], Dostupno na: https://github.com/dotnet-architecture/eShopOnWeb, [Pristupljeno 6.6.2020.]
[16] "What is Odoo?", [Na Internetu], Dostupno na: https://www.softwareadvice.com/crm/odoo-profile/, [Pristupljeno 8.6.2020.]
[17] "OpenProject", [Na Internetu], Dostupno na: https://github.com/opf/openproject, [Pristupljeno 9.6.2020.]

Sustav za nadgledanu nadogradnju programskih komponenti

Sažetak

U ovome radu prolazi se kroz postupak izgradnje sustava za nadgledanu nadogradnju programskih komponenti. Ideja rada je stvoriti sustav koji bi omogućio paralelno izvršavanje akcija na različitim verzijama aplikacije u svrhu testiranja jedne od verzija aplikacije koja može biti nadograđena. Kriterij ispravnosti rada nagodrađene verzije aplikacije svodi se na uspoređivanje odgovora na razini aplikacijskog sloja. Ako su razlike odgovora svedene na faktore koji ovise o slučajnim brojevima, tada je rad nadograđene verzije ispravan. Rezultata ovoga rada je proxy sustav koji omogućuje paralelan rad dviju različitih verzija aplikacija te usporedbu njihovih odgovora na konkretne zahtjeve. U radu se nalaze primjeri konkretnog korištenja rada kako bi se pokazala primjena u praksi.

Ključne riječi: nadogradnja programske potpore, mitmproxy, paralelizacija, Docker, web aplikacija, JSON, HTML, HTTP zahtjev, HTTP odgovor

System for a controlled update of software components

Abstract

This paper goes through the process of building a system for a controlled update of software components. The main idea of the process is to create a system that would allow parallel execution of actions on different versions an application. This way the upgraded version can be tested for bugs and unwanted errors. The criterion for the correct operation of the upgraded version of the application comes down to comparing the responses at the application layer level. If the differences in response are reduced to factors that depend on random numbers, then the operation of the upgraded version is correct. The result of this paper is a proxy system that allows the parallel operation of two different versions of applications and a comparison of their responses. The paper provides examples of concrete usage of the system to demonstrate its application in practice.

Keywords: software patching, mitmproxy, parallelization, Docker, web application, JSON, HTML, HTTP request, HTTP response