Izrada programa za visoku dostupnost servera u trenutku pojave neželjenih događaja

Bister, David

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
STRUČNI STUDIJ RAČUNARSTVO

DAVID BISTER

IZRADA PROGRAMA ZA VISOKU DOSTUPNOST SERVERA U TRENUTKU POJAVE NEŽELJENIH DOGAĐAJA

ZAVRŠNI RAD

Čakovec, rujan 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
STRUČNI STUDIJ RAČUNARSTVO

DAVID BISTER

DEVELOPMENT OF A PROGRAM FOR HIGH AVAILABILITY OF SERVERS AT THE TIME OF OCCURRENCE OF ADVERSE EVENTS

ZAVRŠNI RAD

Mentor:
mr. sc. Željko Knok, v. pred.

Čakovec, rujan 2022.

MEĐIMURSKO VELEUČILIŠTE U ČAKOVCU
ODBOR ZA ZAVRŠNI RAD

Čakovec, 28. veljače 2022.

država:	**Republika Hrvatska**
Predmet:	**Baze podataka I**
Polje:	**2.09 Računarstvo**

ZAVRŠNI ZADATAK br. 2021-RAC-R-39

Pristupnik:	**David Bister (0152211889)**
Studij:	redovni preddiplomski stručni studij Računarstvo
Smjer:	Programsko inženjerstvo

Zadatak: **Izrada program za visoku dostupnost servera u trenutku pojave nezeljenih dogadaja**

Opis zadatka:

LDS je skracenica od Libelle DataSuite.
Zadatak zavrsnog rada je kreirati novi program koji ce biti zaduzen za osiguravanje visoke dostupnosti severa(LDS-Switch).
Zadaca programa je trenutku pojave nezeljenih dogadjaja, koristeci posebnu mreznu konfiguraciju aktivnog servera prilikom nezeljenih dogadaja drugi sekundarni server automatski konfigurirati,korisnici usluga severa nece niti primijeniti bilo kakve posljedice neželjenog događaja.
Program ce biti napisan u programskom jeziku Python,koristeci Django Web-framework.
Prvi korak mora biti detaljna analiza,drugi funkcijska specifikacija, dok ce sama implementacija biti u trecem koraku izrade.

Zadatak uručen pristupniku: 24. siječnja 2022.
Rok za predaju rada: 20. rujna 2022.

Mentor:

mr. sc. Željko Knok, v. pred.

Predsjednik povjerenstva za završni ispit:

ZAHVALA

Putem ove zahvale htio bih se zahvaliti mentoru mr. sc. Željku Konku na svim savjetima i pruženim znanjima.

Zahvaljujem se i tvrtki Libelle AG na pruženoj prilici za rad i istraživanje na ovu temu.

Također bih se želio zahvaliti svojoj baki, djedu i majci na pruženoj potpori.

SAŽETAK

Ako se u današnjem svijetu, u kojem je gotovo svaka radnja digitalizirana i kad se svemu pristupa *online,* dogodi nešto što bi uzrokovalo prekid dostupnosti podataka, bio on planiran ili neplaniran, u velikoj većini slučajeva desio bi se kaos, veliki financijski gubici i nepovjerenje prema različitim kompanijama. Podaci se spremaju u baze podataka i kako bi se spriječio bilo kakav prekid njihove dostupnosti, potrebne su sigurnosne kopije tih baza. Sigurnosne kopije mogu se postići tehnikom zrcaljenja i tehnikom replikacije baze podataka.

Kompanija Libelle AG se kroz svoj proizvod Libelle BussniesShadow bavi automatizacijom postupka zrcaljenja baze podataka, no daljnji razvoj tog proizvoda je otežan jer je baziran na programskom jeziku Lua čije je održavanje zahtjevno. Tako je spomenuta kompanija u svrhu modernizacije svojih proizvoda i prilagodbe tržištu kreirala novi proizvod Libelle DataSuite koji će objediniti sve proizvode Libelle AG. To će omogućiti krajnjim korisnicima kupovine licence za ostale proizvode kompanije čija će se naplata vršiti po *"Pay as you use"* modelu naplate. Novi proizvod za *backend* aplikacije koristi Djanago *framework* i PostgreSQL bazu, dok je frontend aplikacije izrađen u JavaScript biblioteci React. Programski jezik Python nudi veliku podršku zajednice te ima mnoštvo gotovih paketa spremnih za korištenje, a njegov Django *framework* nudi jednostavnost izrade web-aplikacija.

Cilj ovog rada je u Libelle DataSuite integrirati novi proizvod koji će prilikom tehnike zrcaljenja baze podataka u trenutku kad se javi potreba za mijenjanje uloge zrcaljene baze podataka i aktivne baze podataka, kroz frontend proizvoda omogućiti prijavljenom administratoru da jednostavnim postupkom zrcaljenu bazu podataka učini aktivnom i dostupnom na mreži.

U pisanome dijelu rada bit će opisane tehnologije koje se koriste za integraciju novog proizvoda u Libelle DataSuite, postupci sigurnosnih kopija baza podatka, opis proizvoda Libelle DataSuite, analiza za postupak izrade proizvoda, pregled funkcionalisti proizvoda, analiza funkcionalnosti proizvoda, te u zaključku stvari koje se mogu poboljšati.

Ključne riječi: Django framework, React, visoka dostupnost, Material Ui, PostgreSql, Libelle DataSuite

Sadržaj

1. UVOD

Informacije i podaci danas su od neizmjerne važnosti različitim kompanijama. Oni se spremaju u baze podataka, no ako dođe do različitih faktora kojima se uzrokuje gubitak podataka iz baze, to stvara velike financijske troškove za kompaniju, različite probleme unutar strukture kompanije i gubitak povjerenja krajnjih korisnika. Kako bi se izbjegao gubitak podataka, stvaraju se kopije baze podataka. Razlikujemo dvije tehnike kopije baze podataka: zrcaljenje baze podataka i replikacija baze podataka. Za postupak zrcaljenja baze podataka nude se mnoga automatizirana rješenja. Jedno od takvih rješenja nudi kompanija Libelle AG sa svojim dosadašnjim proizvodom Libelle BusinessShadow, te je u svrhu modernizacije i prilagodbe za današnje tržište odlučila kreirati novi proizvod Libelle DataSuite koji će biti osnovna aplikacija za krajnjeg korisnika, koji će po potrebi moći instalirati i koristiti ostale proizvode kompanije, nakon kupljene licence za ostale proizvode čija će se naplata vršiti po principu *Pay-as-you-use* modela naplate.

U ovome zadatku fokusirat će se na postojeći proizvod i kako u njegov dio HA, koji priprema zrcaljenu bazu podataka da postane aktivna, naći programsko rješenje koje će novu aktivnu bazu učiniti dostupnom krajnjim korisnicima u trenutku kad se za to javi potreba, odnosno kad se administrator sustava odluči za to.

Programsko rješenje neće predstavljati finalno rješenje zbog trenutnih nedostataka postojećeg proizvoda Libelle DataSuite koji će nakon perioda realizacije u potpunosti zamijeniti zastarjele proizvode kompanije Libelle AG.

2. KORIŠTENE TEHNOLOGIJE I ALATI

2.1 Django *framework*

Django je Python *web framework* visoke razine koji omogućuje brz razvoj sigurnih web-stranica koje se mogu lako održavati.[1]

Django koristi MVT (*Model-View-Template*) arhitektonski obrazac za razvoj web-aplikacija.

Model upravlja podacima i predstavljen je bazom podataka. Model je u osnovi tablica baze podataka.[2]

View prima HTTP zahtjeve i šalje HTTP odgovore. *View* je u interakciji s modelom i predloškom kako bi dovršio odgovor.[2]

Template je u osnovi front-end sloj i dinamička HTML komponenta Django aplikacije.[2]

Za korištenje Djanga potrebno je imati instalirani Python, te kroz sustav za instaliranje paketa za Python instalirati Django.

Nakon instalacije Django paketa potrebno je iz sučelja naredbenog retka navigirati se u direktorij u kojem se želi izvršiti kreiranje Django aplikacije i upisati naredbu „django-admin startproject imeprojekta“, te se za pokretanje kreirane aplikacije izvršiti naredbu „python manage.py runserver“.

2.2 React

React je deklarativna, učinkovita i fleksibilna JavaScript biblioteka za izgradnju korisničkih sučelja. Omogućuje sastavljanje složenih korisničkih sučelja od malih i izoliranih dijelova koda koji se nazivaju "komponente". [3]

Za razvoj projekta koji koristi React potreban je Node.js.

Node.js je radno okruženje otvorenog koda za više platformi za razvoj poslužiteljskih i mrežnih aplikacija[4]. Nakon instalacije radnog okruženja potrebno je sučelja naredbenog retka navigirati u direktorij u kojem se želi izvršiti kreiranje React aplikacije i upisati naredbu: „npx create-react-app imeaplikacije“, navigirati se u stvorenu aplikaciju te izvršiti naredbu „npm run“.

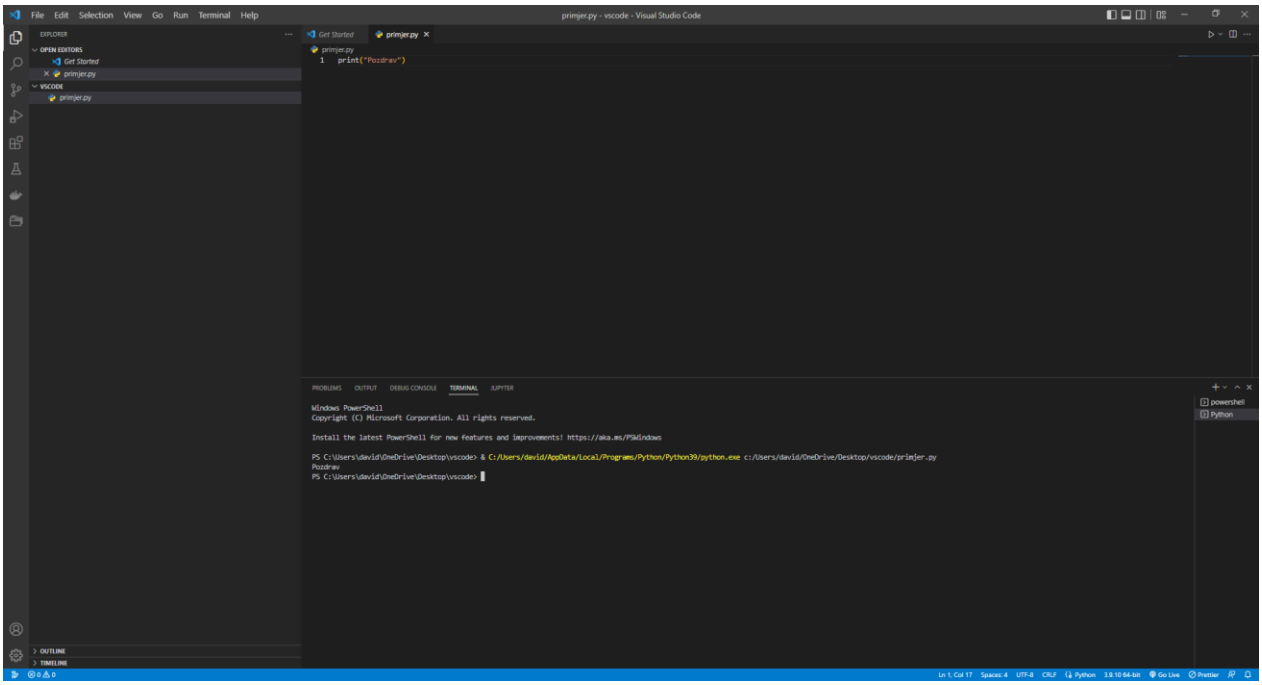
2.3 Material UI

Material UI je biblioteka gotovih React komponenti otvorenog koda koja implementira Googleov materijalni dizajn[5]. Dizajn komponenti je standardiziran, no nudi opcije prilagodbe dizajna komponenata po potrebi. Najveća prednost koju Material UI omogućuje je brzina kojom se može razviti projekt.

Za korištenje biblioteke u React projektu potrebno je kroz sučelje naredbenog retka izvršiti sljedeću naredbu: „npm install @mui/material @emotion/react @emotion/styled“.

2.4 Visual Studio Code

Visual Studio Code je uređivač izvornog koda koji je dostupan za MacOS, Linux i Windows operativne sustave. Nudi razna proširenja koja omogućuju veću produktivnost rada, s integriranim IntelliSensom omogućava značajke koje uključuju dovršavanje sintakse koda, informacije o parametrima i mnoge druge značajke. Spada u skupinu najpopularnijih uređivača izvornog koda, sami uređivač je dostupan u web-verziji.[6]



Slika 1. Izgled Visual Studio Code editora

Izvor: autor

2.5 PgAdmin 4

PgAdmin 4 je alat za upravljanje otvorenim kodom za PostgreSQL, najnapredniju bazu podataka otvorenog koda na svijetu. PgAdmin 4 je dizajniran kako bi zadovoljio potrebe početnika i iskusnih Postgres korisnika, pružajući moćno grafičko sučelje koje pojednostavljuje stvaranje, održavanje i korištenje objekata baze podataka.[7]
Ugrađeni ERD *tool* omogućava dizajn baze podataka koji pruža grafički prikaz tablica baze podataka, stupaca i međuodnosa.[8]

3. BAZE PODATAKA

Baza podataka je organizirana zbirka strukturiranih informacija ili podataka, obično elektronički pohranjenih u računalnom sustavu. Bazom podataka obično upravlja sustav za upravljanje bazom podataka (DBMS). Podaci i DBMS, zajedno s aplikacijama koje su s njima povezane, nazivaju se sustavom baze podataka, često skraćeno na samo baza podataka. Podaci unutar najčešćih vrsta baza podataka koje danas rade obično su modelirani u redovima i stupcima u nizu tablica kako bi obrada i upiti podataka bili učinkoviti.[9]

Podacima se tada može lako pristupiti, njima se može upravljati, mijenjati, ažurirati, kontrolirati i organizirati. Većina baza podataka koristi strukturirani jezik upita (SQL) za pisanje i postavljanje upita podacima.[9]

3.1 Vrste baza podataka

Relacijske baze podataka postale su dominantne 1980-ih. Stavke u relacijskoj bazi podataka organizirane su kao niz tablica sa stupcima i redovima. Tehnologija relacijske baze podataka pruža najučinkovitiji i najfleksibilniji način pristupa strukturiranim informacijama.[9]

NoSQL ili nerelacijska baza podataka omogućuje pohranjivanje i rukovanje nestrukturiranim i polustrukturiranim podacima (za razliku od relacijske baze podataka, koja definira kako moraju biti sastavljeni svi podaci umetnuti u bazu podataka). NoSQL baze podataka postale su popularne kako su web-aplikacije postajale sve češće i složenije.[9]

Odabir odgovarajuće vrste baza podataka ovisi o različitim zahtjevima kako se podaci namjeravaju koristi.

3.2 Zrcaljenje baze podataka

Zrcaljenje (eng. *Mirroring*) je proces generiranja višestrukih kopija baza podataka, a poznat je i kao sjenčanje (eng. *Shadowing).* Ove kopije baze podataka obično se nalaze na drugom računalu. Ako se bilo koji primarni poslužitelj sruši ili angažira na održavanju, u tom trenutku sustav se može automatski prebaciti na zrcaljenu bazu podataka. U bilo kojem trenutku može se pristupiti samo jednoj kopiji.[10]

Čvrsta veza između primarne baze podataka i zrcaljene baze podataka uspostavlja se uz pomoć slanja blokova dnevnika transakcija (eng. *Transaction log)* u zrcaljenu bazu podataka. U slučaju bilo kakvog kvara, također je sposoban vratiti podatke kopiranjem iz jedne baze podataka u drugu. Kada se dogodi bilo koji *failover*, zrcalna baza podataka postaje glavna baza podataka. Zrcaljenje uključuje ponavljanje operacija ažuriranja, umetanja i brisanja koje su izvršene u primarnoj bazi podataka u zrcaljenoj bazi podataka bez ikakvog odgađanja. [10]

U potpuno sigurnom načinu rada, transakcija se ne može izvršiti sve dok zapisi dnevnika za transakciju ne dospiju na disk na zrcalu. Zrcaljenje ne podržava distribuiranu bazu podataka.[10]

3.3 Replikacija baze podataka

Replikacija je proces stvaranja distribucije suvišnih podataka i objekata baze podataka u različitim bazama podataka kako bi se poboljšala dostupnost podataka. Sposoban je skupljati korporativne podatke s geografski raspršenih stranica i širiti podatke udaljenim korisnicima na lokalnim mrežama ili internetu. Povećava izvršavanje paralelnih naredbi.[10]

3.4 Razlike između zrcaljenja i replikacije baze podataka

Ključne razlike između zrcaljenja i replikacije baze podataka:

1. Zrcaljenje uključuje dupliciranje baze podataka pohranjene na različitim strojevima gdje je izvorna baza podataka poznata kao primarna baza podataka, a kopirana baza podataka poznata je kao zrcalna baza podataka. S druge strane, replikacija je dupliciranje podataka i objekata baze podataka pohranjenih na različitim lokacijama radi poboljšanja performansi distribucijske baze podataka.[10]

2. Zrcaljenje se izvodi na bazi podataka dok se replikacija implementira na podatke i objekte baze podataka.[10]

3. Zrcalna baza podataka obično se može pronaći na drugom računalu od svoje primarne baze podataka. Nasuprot tome, replicirani podaci i objekti baze podataka pohranjuju se u drugu bazu podataka.[10]

4. Zrcaljenje baze podataka košta više od replikacije.[10]

5. Zrcaljenje ne podržava distribuirano okruženje, dok je replikacija osmišljena za distribuiranu bazu podataka.[10]

3.5 Što visoka dostupnost baze podataka zapravo znači ?

Visoka dostupnost (eng. *High availability*) je karakteristika sustava koja ima za cilj osigurati dogovorenu razinu operativnih performansi, obično neprekidnog rada, za razdoblje dulje od normalnog.[11]

Vrijeme rada i dostupnost općenito se koriste kao sinonimi. Kako bi se postigla visoka dostupnost i održalo dogovoreno vrijeme rada, arhitekti se pobrinu da smanje prekide rada.[11]

Prekidi usluge dolaze u dvije glavne vrste, a to su planirani prekid rada i neplanirani prekid rad.

Planirani prekid rada – rezultat je aktivnosti održavanja koje ometaju rad sustava i obično se ne mogu izbjeći. Može uključivati zakrpe za sistemski softver koje zahtijevaju ponovno pokretanje sustava ili ponovno pokretanje baze podataka. Općenito, planirani prekid je rezultat nekog logičnog događaja koji je pokrenuo menadžment.[11]

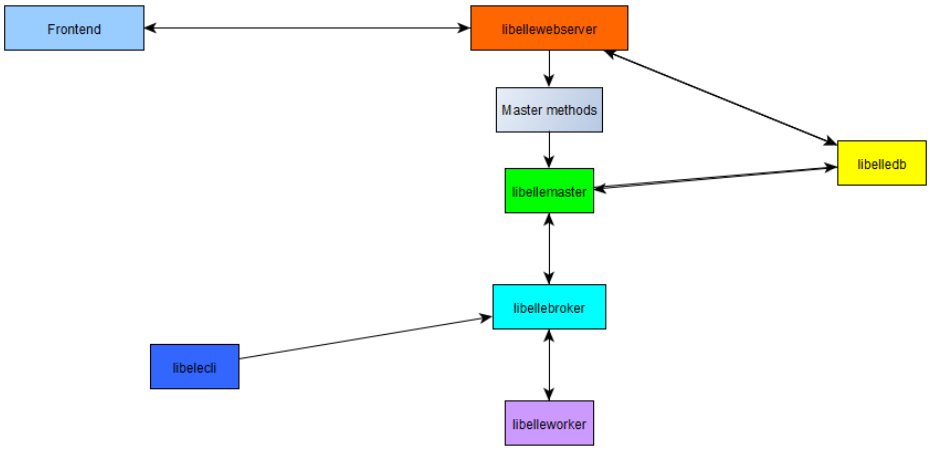
Neplanirani prekid rada – rezultat je događaja prekida rada zbog nekih fizičkih kvarova/događaja, kao što je kvar hardvera ili softvera ili ekološka anomalija. Na primjer, nestanci struje, neispravne komponente središnje jedinice za obradu *(eng. Central processing unit)* ili memorije s nasumičnim pristupom (eng. *Random Access Memory*) te drugi mogući kvar drugih hardverskih komponenti, kvar mreže, narušavanje sigurnosti ili kvarovi raznih aplikacija, međuprograma i operativnog sustava rezultiraju neplaniranim ispadom/neplaniranim zastojem.[11]

4. OPIS POSTOJEĆEG STANJA

LDS (Libelle DataSuite) je proizvod nove generacije kompanije Libelle AG. Proizvod za sada nije dostupan javnosti jer je u fazi razvoja i promoviranja. Razvija se u modernim tehnologijama kao što su Django i React. Inicijalna ideja je pojednostaviti postupke instalacija i izdavanja licenci za proizvode te omogućiti upravljanje frontendom preko svih uređaja (pametni telefoni, računala, tableti). Za korištenje proizvoda uglavnom su potrebna dva računala koja čine distribuirani sustav.

Sukladno s integracijom novih proizvoda u LDS će se dodavati značajke potrebne za implementaciju novih proizvoda.

4.1 Arhitektura LDS-a



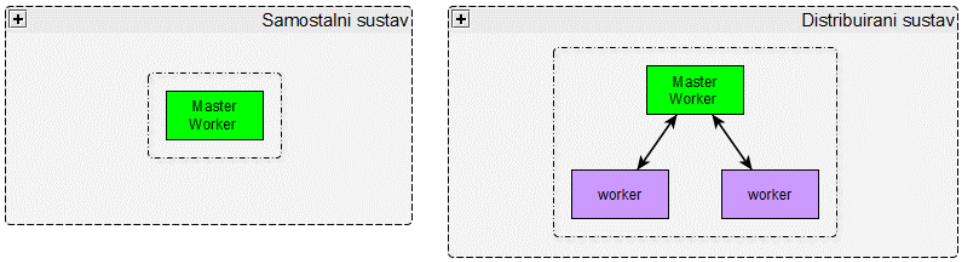
Slika 2. Arhitektura LDS-a

Izvor: autor

Sama arhitektura LDS-a koristi arhitektonski uzorak *master-broker-worker*, odnosno sa samog frontenda može se pokrenuti određeni zadatak. Ovisno o zadatku, pokreću se ili master metode ili se vrše zahtjevi za podacima iz baze. Ako se pokreću master metode, *libellewebserver* pokreće master metodu za određeni zadatak. Ona generira zadatak te ga prosljeđuje *libellemaster* komponenti. Nakon toga komponenta šalje poruku *libellebrokeru* da se izvrši određeni zadatak na određenom *workeru*, nakon primljenog zadatka *libelleworker* izvršava zadatak te vraća

rezultat zadatka *libellebrokeru* te on sukladno tome vraća rezultat *libellemasteru* koji poruku o uspješnosti zadatka ili sami rezultat zapisuje u *libelledb* (bazu podataka). Svaki zadatak koje master metode generiraju je u JSON formatu.

Sama moć arhitekture je u tome što možemo postići samostalni sustav (eng. *single system*), te također distribuirani sustav (eng. *distributed system*). Za potrebe ovog završnog rada koristit će se distribuirani sustav.



Slika 3. Prikaz samostalnog sustava te distribuiranog sustava

Izvor: autor

Također postoji *libellecli* komponenta (Slika 2) koja nema potpunu funkcionalnost *libellemaster* komponente, već se iz naredbenog retka može izvršiti zahtjev da se izvrši određeni zadatak na jednome *workeru* (samostalni sustav), dok se na *libellemaster* komponenti mogu izvršiti zadaci na distribuiranom sustavu.

4.2 Standardna master instalacija

Za samu instalaciju potreban je instalacijski paket te se preko cmd-a navigirati do paketa i pokrenuti instalaciju s naredbom „.\libellesetup.cmd -interactive“ te nakon toga izabrati konfiguracijske parametre i pokrenuti instalaciju. U ovome slučaju radi se o standardnoj master instalaciji (samostalni sustav).

Slika 4. Standardna master instalacija (samostalni sustav)

Izvor: autor

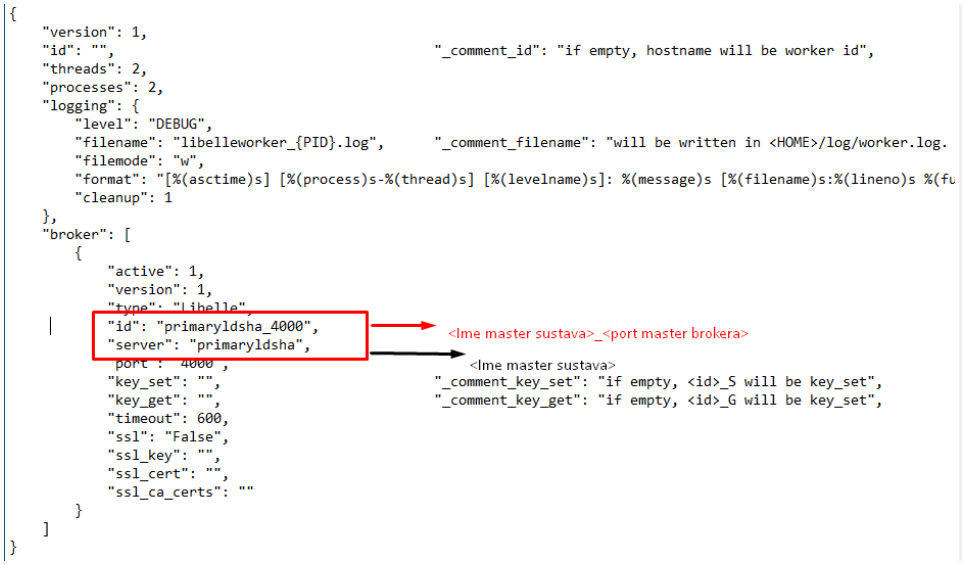
4.3 Prilagođena *worker* instalacija

Kad se želi ostvariti distribuirani sustav, potrebna je *worker* instalacija na sustavu koji je namijenjen kao *worker*. Primijenit će se isti postupak kao u poglavlju 2.1, samo s drugačijim parametrima. Kao tip instalacije odabire se *worker*, te se odabire prilagođena instalacija *(eng. Customized)*, jedina komponenta koja se ne instalira je *libellebroker*, jer će sami *worker* koristiti *broker* od master instalacije.

Slika 5. Prilagođena worker instalacija (distribuirani sustav)

Izvor: autor

Nakon uspješne prilagođene *worker* instalacije, potrebno je na mjestu instalacije „C:\libelle\lds“ urediti datoteku „libelleworker.json“ tako da u ključu „id“ *(eng. Key)* dodamo vrijednost *(eng.Value)* _ u ovome slučaju „primaryldsha_4000“, te još dodati ključu „server“ vrijednost.



Slika 6. Uređivanje datoteke libelleworker.json

Izvor: autor

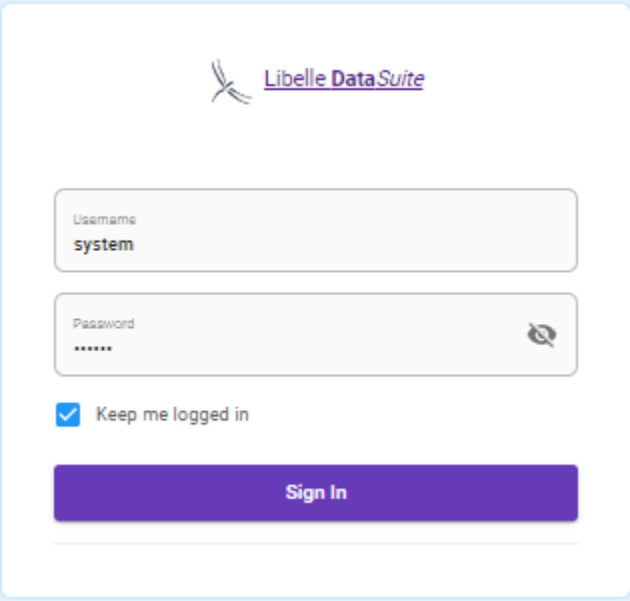
Nakon završetka uređivanja datoteke potrebno je ponovo pokrenuti *libelleworker* komponentu. To se može postići tako da se u preglednik upiše http://localhost:<LibelleSupervisor port>/ i pritiskom na restart na *libelleworker* komponenti.

Slika 7. Ponovo pokretanje libelleworker komponente

Izvor: autor

4.4 Prijava korisnika u LDS-u

Prvi korak je pokretanje frontenda postojećeg programa Libelle DataSuite. Ako se koristi standardni port za *libellewebserver* komponentu, potrebno je u web-pregledniku upisati https://localhost:8000/, trenutno postoji jedna razina prijave kao administrator koji ima sva prava kojima može upravljati svim konfiguracijama.

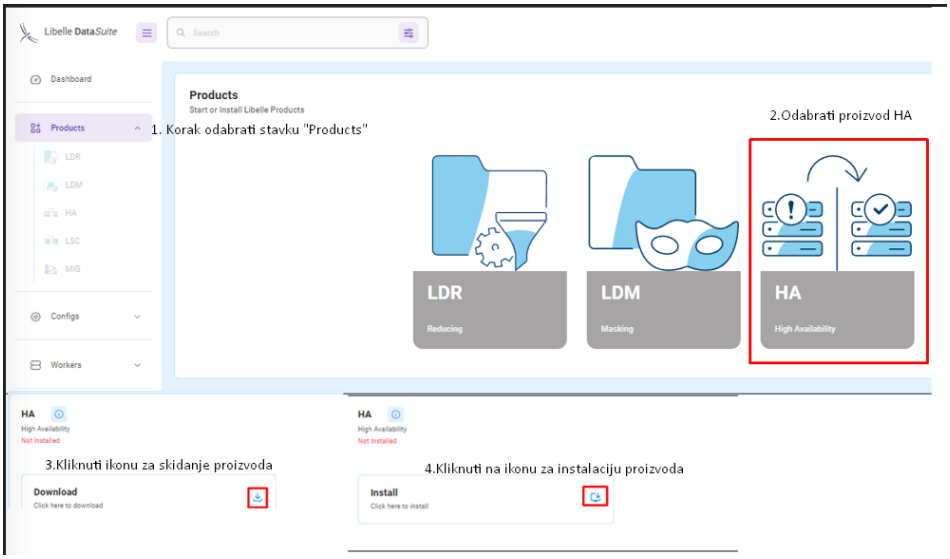


Slika 8. Prijava u Libelle Data Suite kao administrator

Izvor: autor

4.5 Instalacija proizvoda u LDS-u

Nakon prijave u sustav potrebno je odabrati stavku u "Products" i odabrati proizvod koji se želi instalirati, u ovome slučaju proizvod "HA". Nakon odabira proizvoda otvorit će se izbornik koji nudi opciju preuzimanja proizvoda klikom na ikonu proizvoda. Nakon preuzetog proizvoda nudi se opcija za instalaciju te se klikom na instalacijsku ikonu proizvoda pokrene instalacija.



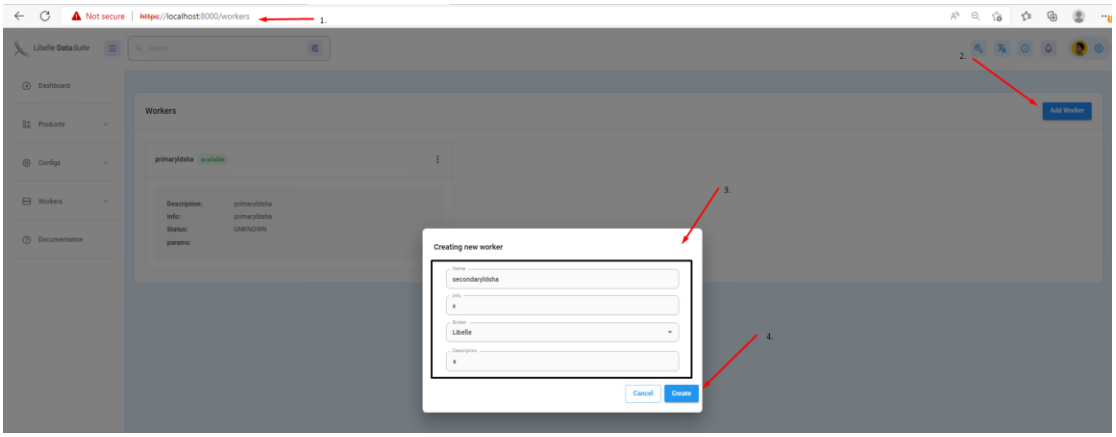
Slika 9. Opis instalacije proizvoda HA

Izvor: autor

Kao finalan korak potrebno je na frontendu standardne master instalacije dodati prethodno konfigurirani *worker*.

Sami postupak dodavanja *workera* dijeli se na nekoliko koraka:

1. Pokretanje frontenda u pregledniku http://localhost:8000/workers
2. Klik na gumb „Add Worker“
3. Popunjavanje podataka o *workeru*
4. Klik na gumb „Create“.



Slika 10. Postupak dodavanja workera

Izvor: autor

Nakon uspješnog dodavanja *workera*, distribuirani sustav spreman je za korištenje.

4.6 Gdje se zapravo odvija razvoj frontenda ako je on nedostupan u sustavu LDS-a?

Instalacija proizvoda kroz frontend omogućava razvojnim programerima instalaciju proizvoda na testnim okruženjima. Proizvod koji je u razvoju, trenutno proizvod "HA" koji nije spreman za generalno testiranje od strane tima za testiranje, razvija se na zasebnom frontendu, te se po dovršetku proizvoda integrira u glavni frontend LDS-a.

Iako se pokreće nezavisan frontend, potrebno se je prijaviti na frontend LDS-a kako bi osigurali autentifikaciju.

4.7 Struktura proizvoda LDS-a u Django *frameworku*

Svaki proizvod u LDS-u ima istu strukturu jer koriste zajedničke *libelleworker*, *libellemaster* komponente, unutar mape proizvoda src\libelle\products\ha nalazi se podmapa *worker* u kojoj se nalaze datoteke task_ha.py i task_switch.py gdje svaka od njih sadrži logiku proizvoda.

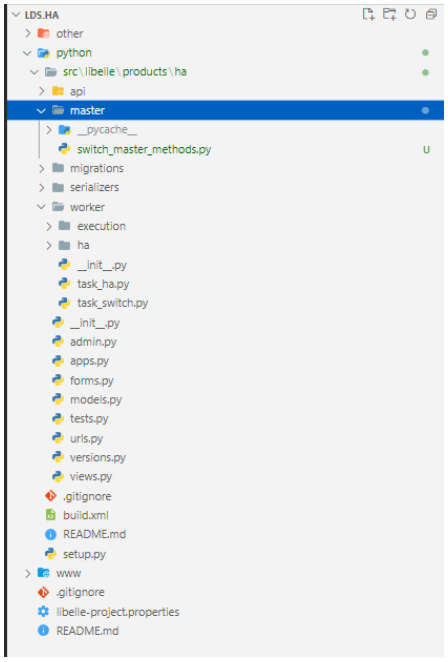
Primjer: u task_ha.py se nalazi logika koja prilikom dobitka zadatka od *libellemaster* komponente izvršava razne zadaće koje omogućavaju proces zrcaljenja baze podataka.

U task_switch.py nalazit će se logika koja prilikom dobitka zadatka od *libellemaster* komponente, da po potrebi kad task_ha.py pripremi zrcaljenu bazu podataka bude aktivna baza podataka, učini zrcaljenu bazu podatka dostupnom na mreži.

Podmapa master je mapa u kojoj se nalaze master metode. One su zaslužne za kreiranje zadatka kojim se pokreće proces izvršavanja zadataka na *libellemaster* komponenti.

Podmapa api služi za sve datoteke koje omogućuju realizaciju aplikacijskog programskog sučelja (eng. *Application programming interface*).

Mapa www služi za kompletni programski kod za frontend aplikacije.

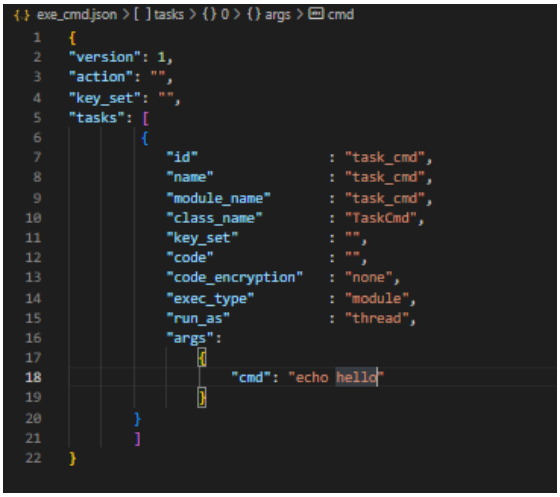


Slika 11. Struktura projekta proizvoda HA

Izvor: autor

4.8 Izvršavanje zadatka na LDS-u

Za ručno izvršavanje potrebno je generirati zadatak za LDS i izvršiti ga na *libellecli* komponenti.



Slika 12. Generiranje zadatka za LDS

Izvor: autor

Za izvršavanje na *libellecli* komponenti potrebno je u libellecli.cmd proslijediti argumente „-execute“ te putanju do određenog zadataka u json formatu (slika 13, oznaka 2). Nakon izvršavanja zadatka ispiše se rezultat zadatka (slika 13, oznaka 1).

Slika 13. Postupak izvršavanja zadatka i rezultat izvršenog zadatka

Izvor: autor

5. CILJEVI ZADATKA

U trenutku planiranih prekida rada, za baze podataka koje svoju visoku dostupnost poslužitelja održavaju postupkom zrcaljenja i omogućuju neprimjetnu izmjenu izvorne baze podataka i zrcaljene baze podataka, krajnjim korisnicima bit će dostupna zrcaljena baza podataka te neće primijetiti ikakvu radnju koja je uzrokovala planirani prekid rada.

Kako bi se uklonila bilo kakva konfuzija, u daljnjem opisu ciljeva zadatka uvest će se pojmovi primarnog sustava, sekundarnog sustava i aktivnog sustava.

Pod pojmom aktivnog sustava podrazumijeva se poslužitelj baze podataka na kojem korisnici mogu vršiti upite i zapisivati. Kad se vrši promjena aktivnog sustava, aktivnim sustavom mogu postati primarni poslužitelj, sekundarni poslužitelj ili ništa (odnosno kad ni primarni ni sekundarni poslužitelj baze podataka nije dostupan). Nadalje će se u ovom radu koristiti samo ti pojmovi.

Glavna ideja dostupnosti baze podataka nakon promjene aktivnog sustava je u korištenju jedne IP adrese za primarni i sekundarni sustav. Takva IP adresa koja ne odgovara stvarnom stanju na fizičkom mrežnom sučelju naziva se virtualna IP adresa.

Primarni i sekundarni sustavi imaju vlastite IP adrese koje odgovaraju njihovome fizičkom mrežnom sučelju, no krajnji korisnici posjeduju samo virtualnu IP adresu te kad primarni ili sekundarni sustav postanu aktivnim sustavom, na njih se dodijeli virtualna IP adresa. Kako bi se izbjegle bilo kakve kolizije, virtualna IP adresa ne smije se nalaziti na primarnom i sekundarnom sustavu istovremeno.

Cilj zadatka je automatizirati postupak dodjeljivanja ili uklanjanja virtualne IP adrese prilikom izmjene aktivnog sustava i integrirati ga u postojeći sustav, te prilikom integracije otkriti trenutne nedostatke postojećeg sustava. U zaključku rada predložit će se promjene koje su potrebne na postojećem sustavu, a koje će omogućiti neometani postupak automatiziranja izmjene aktivnog sustava.

Ostvarivanje cilja zadatka provest će se kroz analizu postupka izrade programa, pregled osnovnih funkcionalnosti, analize funkcionalnosti i zaključka.

6. ANALIZA ZA POSTUPAK IZRADE PROGRAMA

Funkcionalnost bi trebala omogućiti slanje zahtjeva i primanje povratne poruke (ping naredba) ovisno o rezultatu metode. Ako je poslužitelj dobio odgovor, treba provjeriti nalazi li se virtualna IP adresa na nekom od poslužitelja za koji je napravljena konfiguracija. Ako se doista nalazi na nekom od konfiguriranih poslužitelja, treba utvrditi na kojem. Kada se utvrdi na kojim sustavima se nalazi, adresa se dodaje na aktivni sustav i, ako je potrebno, adresa se uklanja s neaktivnog sustava. Ako adresa nije pronađena ni na jednom poslužitelju u konfiguraciji, prikazuje se upozorenje da se virtualna IP adresa nalazi na stranom sustavu. Ako se ping metodom ne dobije odgovor, program provjerava koji je sustav aktivan i prema tome dodaje ili uklanja adresu. Ako nema aktivnog sustava, virtualna se IP adresa uklanja iz sustava koji se nalazi u konfiguraciji. Prema ovoj funkcionalnosti kreirat će se algoritam za izvođenje programa.
Provedenom analizom utvrđeno je da nema gotovog paketa za programski jezik Python kojim bismo mogli postići potrebne funkcionalnosti programa. Shodno tome će se u sljedećim

koracima analizirati kako funkcionalnosti postići putem sučelja naredbenog retka. Za Windows distribucije koristit će se Windows PowerShell, dok će se za Linux distribucije koristiti terminal.

6.1 Ručni postupci za potrebne funkcionalnosti programa za Windows distribucije

Za ostvarivanje lakše kasnije manipulacije podacima u budućem programu svi potrebni podaci koji zahtijevaju programsku obradu preuzimat će se u JSON formatu.

Za dohvaćanje liste sa svim mrežnim sučeljima i njihovim parametrima koristit će se sljedeća naredba: „Get-NetIPConfiguration“.

Ako sami ne formatiramo rezultat i pretvorimo ga u JSON format, Windows PowerShell vraća golemi rezultat s puno sadržaja kojeg nema u prvobitnoj naredbi. Stoga će se za formatiranje rezultata u željeni obradivi format koristiti sljedeća PowerShell skripta:

```
$rezultat = Get-NetIPConfiguration | select InterfaceIndex,
IPv4Address, InterfaceAlias, InterfaceDescription, NetAdapter
ForEach( $a in $rezultat ){
 $a.Ipv4Address = $a.Ipv4Address.IpAddress
 $a | Add-Member -type NoteProperty -name Status -value
$a.NetAdapter.Status
 $a.PSObject.Properties.Remove('NetAdapter')
}
$reultat | ConvertTo-Json
```

Kod 1. Windows Powershell skripta za vraćanje podataka u JSON formatu

Izvor: autor

Za dodavanje IP adrese koristi se naredba:

„New-NetIPAddress –IPAddress < željena ip adresa > –InterfaceAlias <ime željenog mrežnog sučelja> –SkipAsSource $True“

Primjer:“ New-NetIPAddress –IPAddress 10.20.20.25 –InterfaceAlias “Ethernet0” –SkipAsSource $True“

Kako bi se uklonila IP adresa, koristi se naredba:

„Get-NetIPAddress -IPAddress <željena ip adresa > | Remove-NetIPAddress -Confirm:$false“

Primjer: „Get-NetIPAddress -IPAddress 10.20.20.25 | Remove-NetIPAddress -Confirm:$false“

Za dobivanje informacije o statusu postojanja IP adrese na mreži koristit će se naredba ping:

„Ping <željeni uređaj /željeni server>“

Primjer: „ping 10.20.20.17“

6.2 Ručni postupci za potrebne funkcionalnosti programa za Linux distribucije

Za dohvaćanje liste sa svim mrežnim sučeljima i njihovim parametrima koristi se naredba:

„ip --json addr show“

Za dodavanje IP adrese koristi se naredba:

„ip addr add < željena ip adresa i subnet maska> br < broadcast adresa željenog sučelja > dev <ime željenog mrežnog sučelja >“

Primjer:“ ip addr add 10.30.11.220/16 brd 10.30.255.255 dev ens160“

Kako bi se uklonila IP adresa, koristi se naredba:

„ip addr del <željena ip adresa i subnet maska> dev < ime željenog mrežnog sučelja>“

Primjer:

„ip addr del 10.30.11.220/16 dev ens160“

6.3 Automatizacija ručnih postupaka za postizanje funkcionalnosti programa

Za automatizaciju ručnih postupaka za postizanje funkcionalnosti programa u programskom jeziku Python koristi se *subprocess* modul koji omogućuje stvaranje novih procesa, spajanje na njihove ulazne/izlazne cijevi (eng. *Pipes*) pogrešaka i dobivanje njihovih povratnih kodova.[4]

Json je knjižnica koja omogućuje manipulaciju nad JSON formatom.[5]

Primjer automatizacije naredbe Windows Powershell skripte za vraćanje podataka s listom mrežnih sučelja u JSON formatu:

```
import subprocess

import json
class Switch:
    def switch_subprocess(self, command):
        try:
            result = subprocess.run(command, shell=True,
                                    capture_output=True, text=True, check=True)
            out_result = result.stdout

            return out_result
        except Exception as e:
            return (str(e))
list_command_1 = "$n = Get-NetIPConfiguration | select InterfaceIndex, IPv4Ad-
dress, InterfaceAlias, InterfaceDescription, NetAdapter;"
list_command_2 = "ForEach( $a in $n ){$a.Ipv4Address =  $a.Ipv4Address.IpAd-
dress;$a \
    | Add-Member -type NoteProperty -name Status -value \
        $a.NetAdapter.Status;$a.PSObject.Properties.Remove('NetAdapter')};"
list_command_3 = "$n | ConvertTo-Json"
ps_command = f'powershell -command "{list_command_1}{list_command_2}{list_com-
mand_3}"'
switch_object=Switch()
result=switch_object.switch_subprocess(ps_command)
result_list=json.loads(result)
ispis=f"Ime mrežnog sučelja:\
    {result_list[0]['InterfaceAlias']}\
         Ip adresa: {result_list[0]['IPv4Address']}"
print(f"{ispis}")
```

Kod 2. Kod vraća prvo ime mrežnog sučelja i njegovu IP adresu

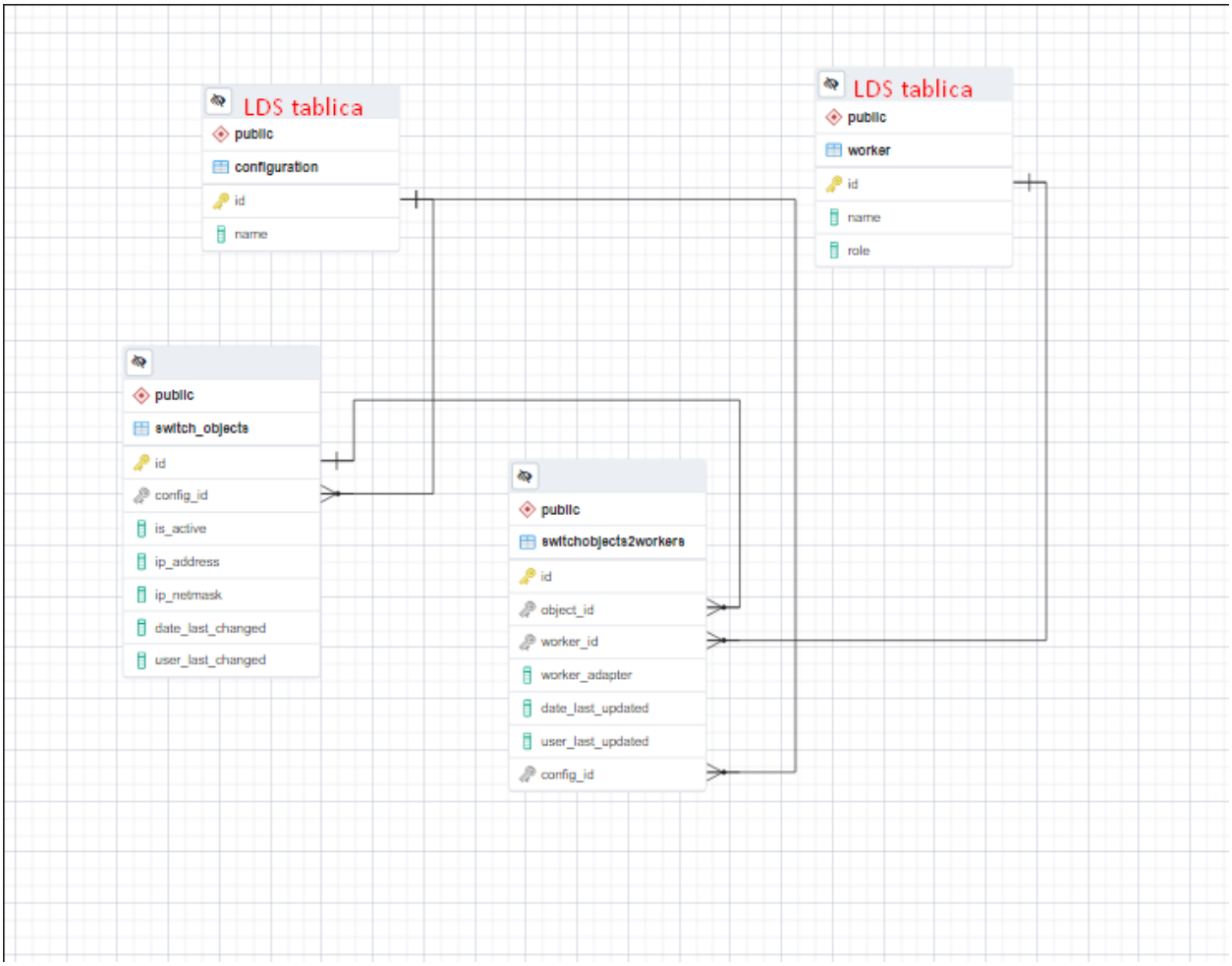
Izvor: autor

6.4 Funkcionalnosti potrebne za frontend program

Funkcionalnosti frontend programa omogućit će krajnjem korisniku da može konfigurirati konfiguraciju za dva sustava (primarni i sekundarni) i njihovu virtualnu IP adresu. Nakon konfigurirane konfiguracije dostupan je meni koji uključuje sve konfiguracije, te poseban meni za određenu konfiguraciju koji će omogućiti pokretanje ili zaustavljanje akcije u *backendu* programa. U glavnome meniju za upravljanje konfiguracijom također bi se trebao nalaziti padajući izbornik koji bi omogućio odabir aktivnog sustava. Korisniku će se također omogućiti prikazivanje poruka o statusu izvršenih funkcionalnosti.

6.5 ER model za bazu podataka

ER model je kratica za *Entity-Relationship* model. To je podatkovni model visoke razine. Ovaj se model koristi za definiranje podatkovnih elemenata i odnosa za određeni sustav. Razvija idejno rješenje baze podataka. Također razvija vrlo jednostavan i za dizajn lak prikaz podataka.[14]



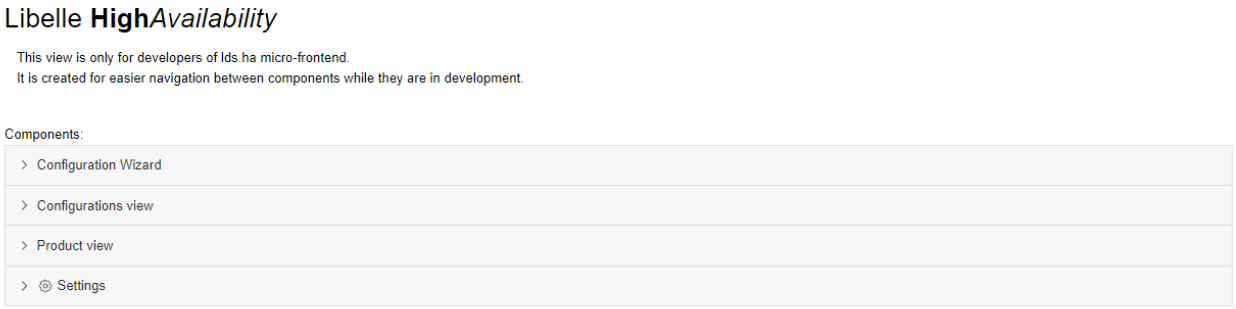
Slika 14. ER model za bazu podatka

Izvor: autor

Ovaj ER model nudi samo osnovnu sliku o izgledu baze podatka koja je potrebna za generalno razumijevanje, te su implementirane samo tablice switch_objects i switchobjects2worker, dok su ostale tablice već integrirane u LDS-u. Za generiranje modela korišten je Erd tool u alatu PgAdmin 4.

7. PREGLED FUNKCIONALNOSTI PROGRAMA

Za pregled funkcionalnosti programa potrebno se je prijaviti na glavnom frontend programu i pokrenuti frontend za razvojne programere.



Slika 15. Izgled stranice za proizvod HA

Izvor: autor

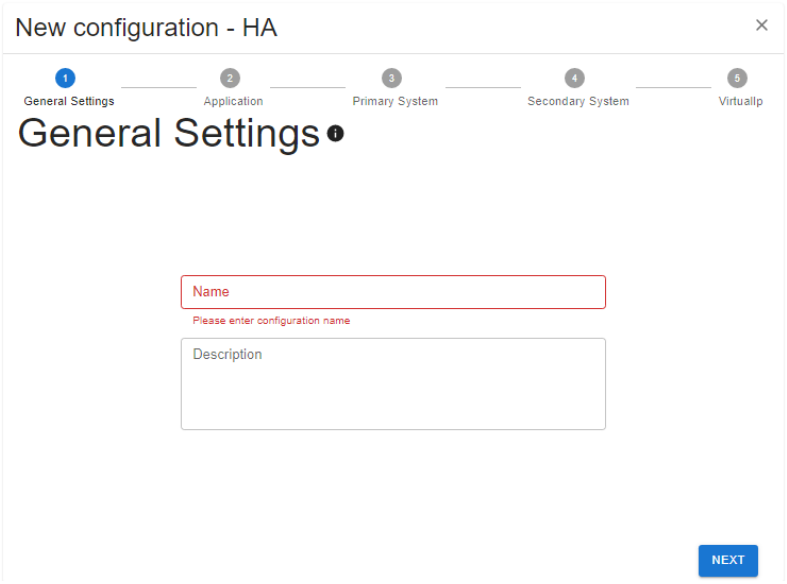
Razmotrit će se sljedeće funkcionalnosti:

1. Kreiranje konfiguracije
2. Pregled kreiranih konfiguracija
3. Pokretanje procesa održavanja visoke dostupnosti poslužitelja baze podataka
4. Promjena aktivnog sustava poslužitelja baze podatka
5. Zaustavljanje procesa održavanja visoke dostupnosti poslužitelja baze podataka.

7.1 Kreiranje konfiguracije

Za kreiranje konfiguracije potrebno je kliknuti na stavku „Configuration wizard“ te nakon toga kliknuti na gumb „Wizard“. Nakon tog klika otvara se Wizard komponenta koja nam omogućava kreiranje konfiguracije, po Wizard komponenti kreće se naprijed klikom na gumb „NEXT“, dok se vraćanje na prethodni korak vrši klikom na gumb „PREVIOUS“. Wizard ima ugrađene mehanizme validacije podataka, a jedini opcionalni dio je unošenje opisa *(eng. Description)* i odabir aplikacije ako opcionalnog dijela nema. Polje za unos imena konfiguracije „ime“ *(eng. Name)* uzima se kao opis.

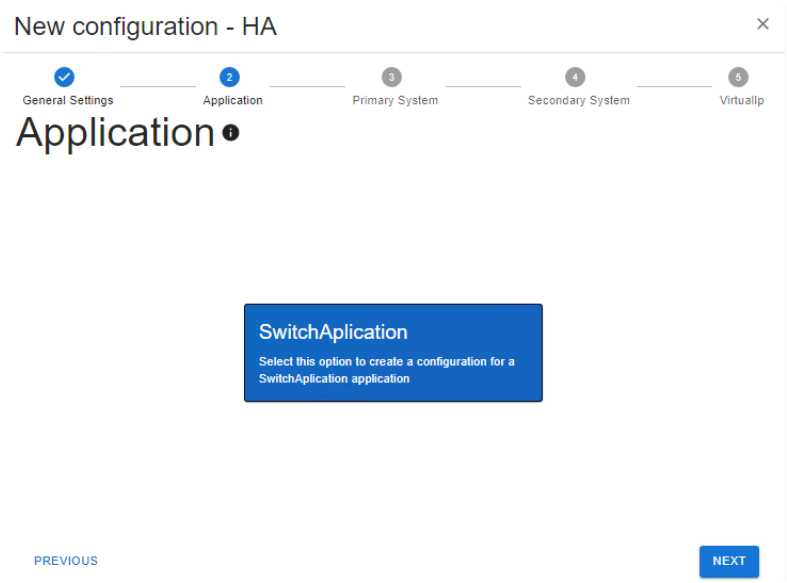
U prvom koraku u Wizard komponenti unosimo ime i opis konfiguracije.



Slika 16. Prikaz prvog koraka u Wizard komponenti

Izvor: autor

U drugom koraku u Wizard komponenti odabiremo aplikaciju te odabiremo gumb „NEXT“.

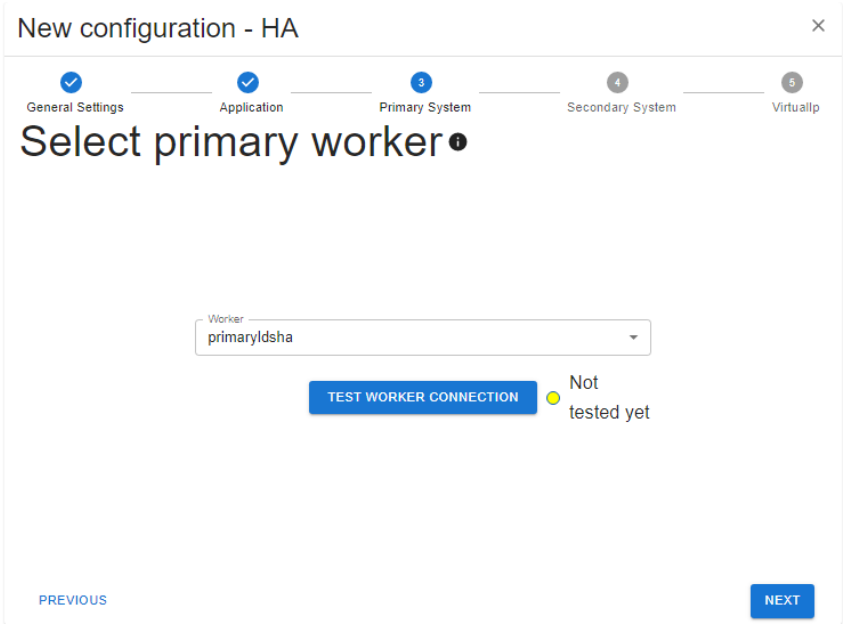


Slika 17. Prikaz drugog koraka u Wizard komponenti

Izvor: autor

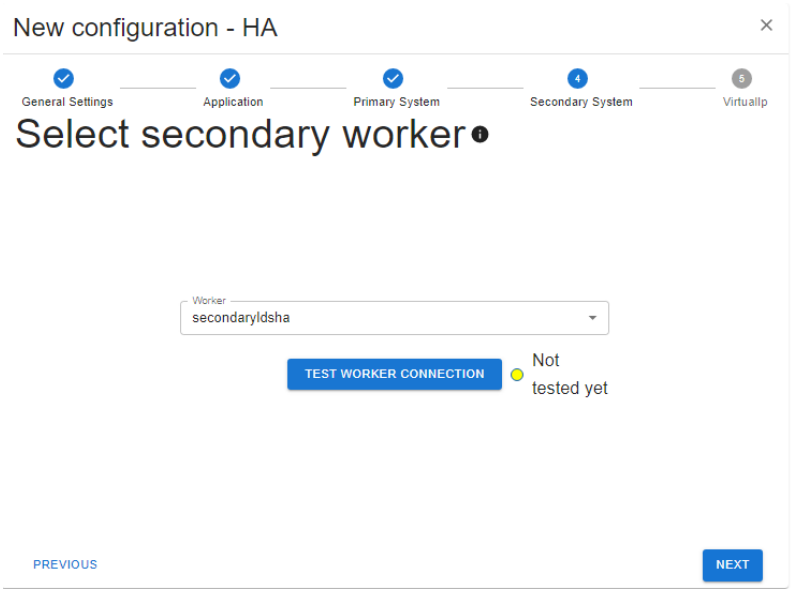
U trećem koraku odabire se primarni sustav gdje se u padajućem izborniku mogu odabrati različiti *workeri* koji su registrirani u sustavu. Ako je korisnik zaboravio registrirati *workera*, u

padajućem izborniku nudi mu se „Add new worker“ opcija koja omogućuje registraciju *workera* u sustavu.



Slika 17. Prikaz trećeg koraka u Wizard komponenti
Izvor: autor

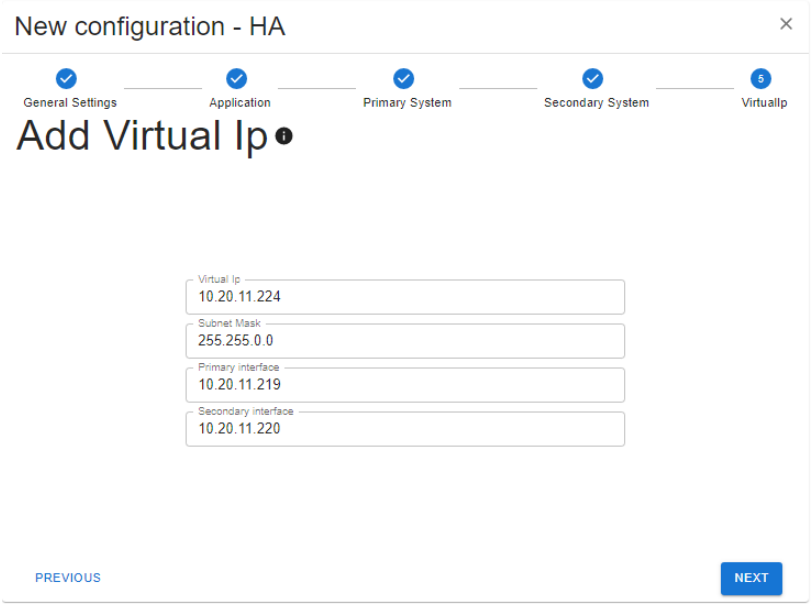
U četvrtom koraku odabiremo sekundarni *worker*, u padajućem izborniku nude se svi *workeri* osim prethodno odabranog.



Slika 18.Prikaz četvertog koraka u Wizard kompnenti

Izvor: autor

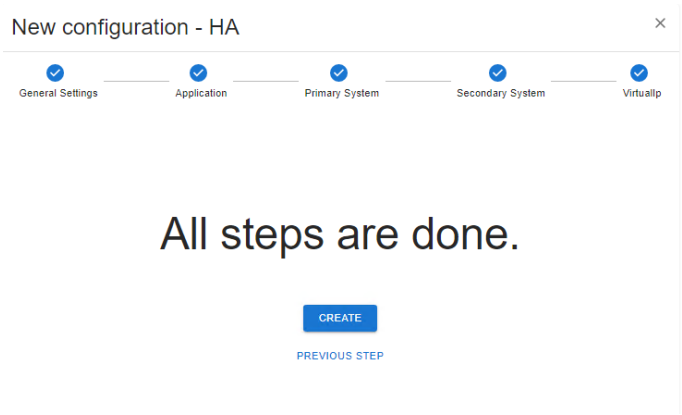
U petom koraku upisuju se podaci o virtualnoj IP adresi, te fizičke adrese mrežnih sučelja za primarni i sekundarni sustav na koje želimo opredijeliti virtualne IP adrese. Validiraju se podaci uneseni za IP adrese, no za provjeru subnet maske koristi se validacija za IP adrese, što znači da ako korisnik unese subnet masku koja ne postoji, podatak će se zapisati u bazu.



Slika 19. Peti korak u Wizard komponenti

Izvor: autor

U šestom koraku nakon ispunjenih svih prethodnih koraka konfiguracije otvara se meni koji omogućava kreiranje konfiguracije ili povratak na prethodne korake. Nakon uspješne kreacije (pritisak na gumb „Create“) pojavljuje se obavijest o uspjehu na gornjem desnom rubu te se korisnika preusmjerava na kreiranu konfiguraciju. Prikazuje se obavijest o uspješnom kreiranju; obavijest se također prikazuje i u slučaju neuspješne konfiguracije.

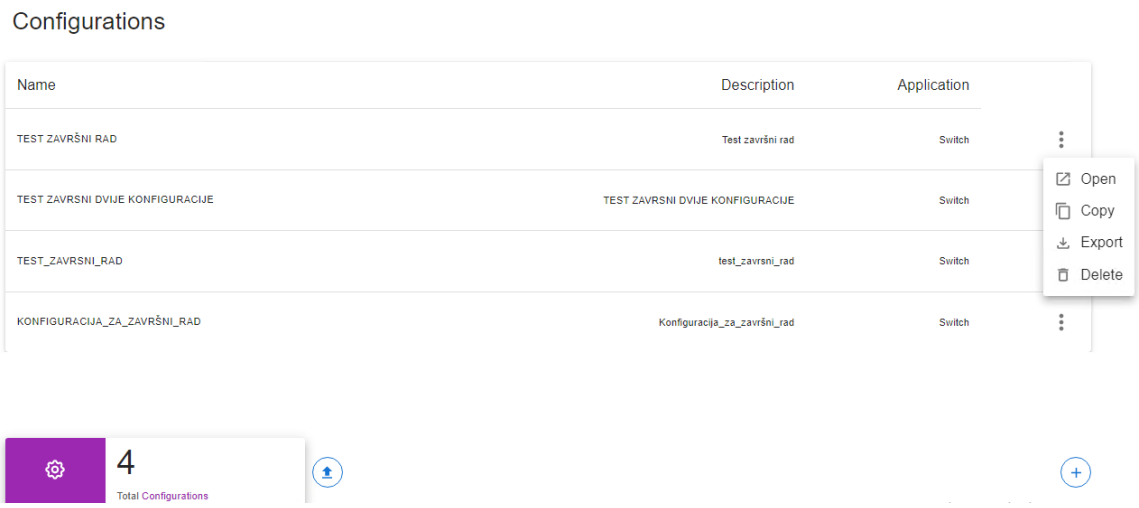


Slika 20. Izbornik za kreiranje konfiguracije u Wizard komponenti

Izvor: autor

7.2 Pregled kreiranih konfiguracija

U komponenti za pregled svih konfiguracija postoje mogućnosti otvaranja, brisanja te izvoza parametra konfiguracije u JSON format.

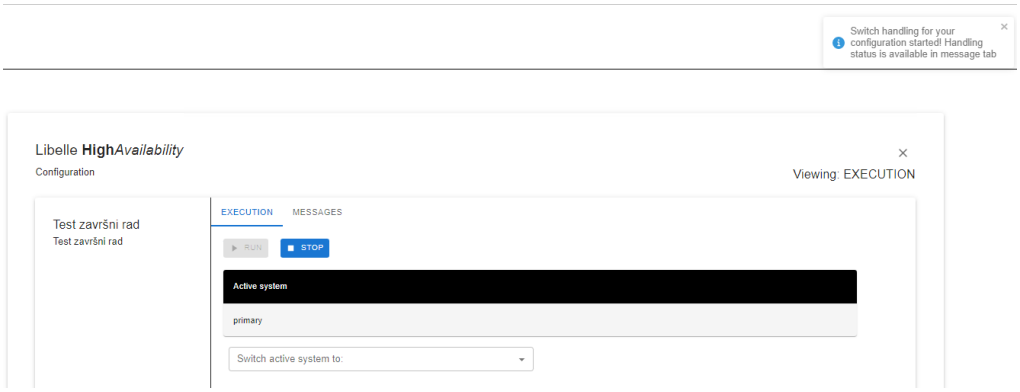


Slika 21. Pregled svih konfiguracija

Izvor: autor

7.3 Upravljanje procesom održavanje visoke dostupnosti poslužitelja baze podataka

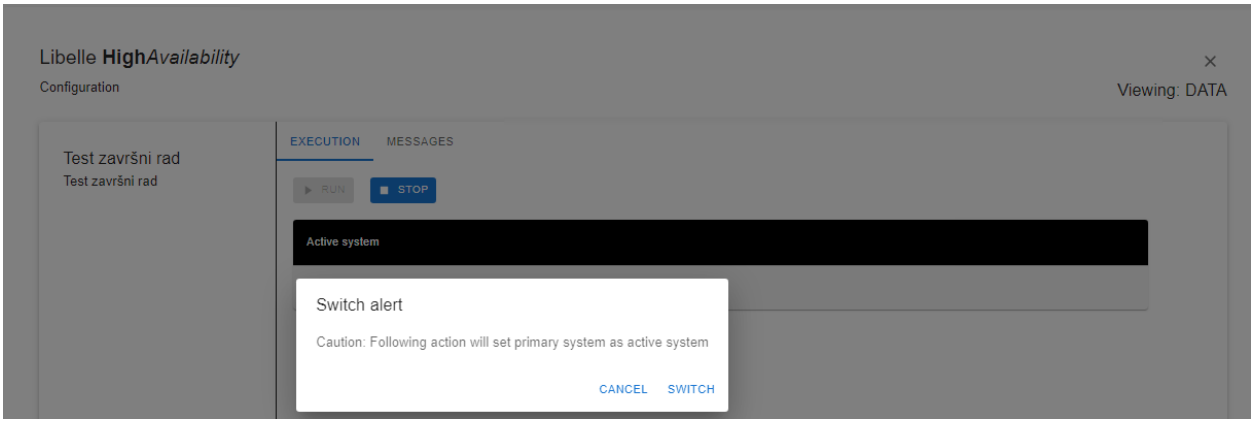
Otvaranje menija za upravljanje konfiguracijom („Execution“) moguće je automatskom redirekcijom nakon kreirane konfiguracije ili otvaranjem kroz pregled konfiguriranih konfiguracija. Klikom na gumb „RUN“ pokreće se proces za održavanje visoke dostupnosti za određenu konfiguraciju, dok se klikom na gumb „STOP“ taj proces zaustavlja. Nakon određene akcije pojavljuje se obavijest o akciji, dok se akcija zapisuje u poruke za određenu konfiguraciju.



Slika 22. Pokretanje procesa održavanje visoke dostupnosti i obavijest o pokretanju

Izvor: autor

Promjena aktivnog sustava vrši se u padajućem izborniku što korisniku omogućuje promjenu aktivnog sustava na primarni i sekundarni sustav. Također se nudi opcija za nijedan aktivni sustav. Nakon odabira stavke pojavljuje se dijalog u kojem se potvrđuje izvođenje određene akcije (klikom na gumb „Switch“) ili se otkazuje (klikom na gumb „Cancel“). Prikazuje se obavijest o izvođenju akcije te se akcija zapisuje u poruke za tu konfiguraciju.

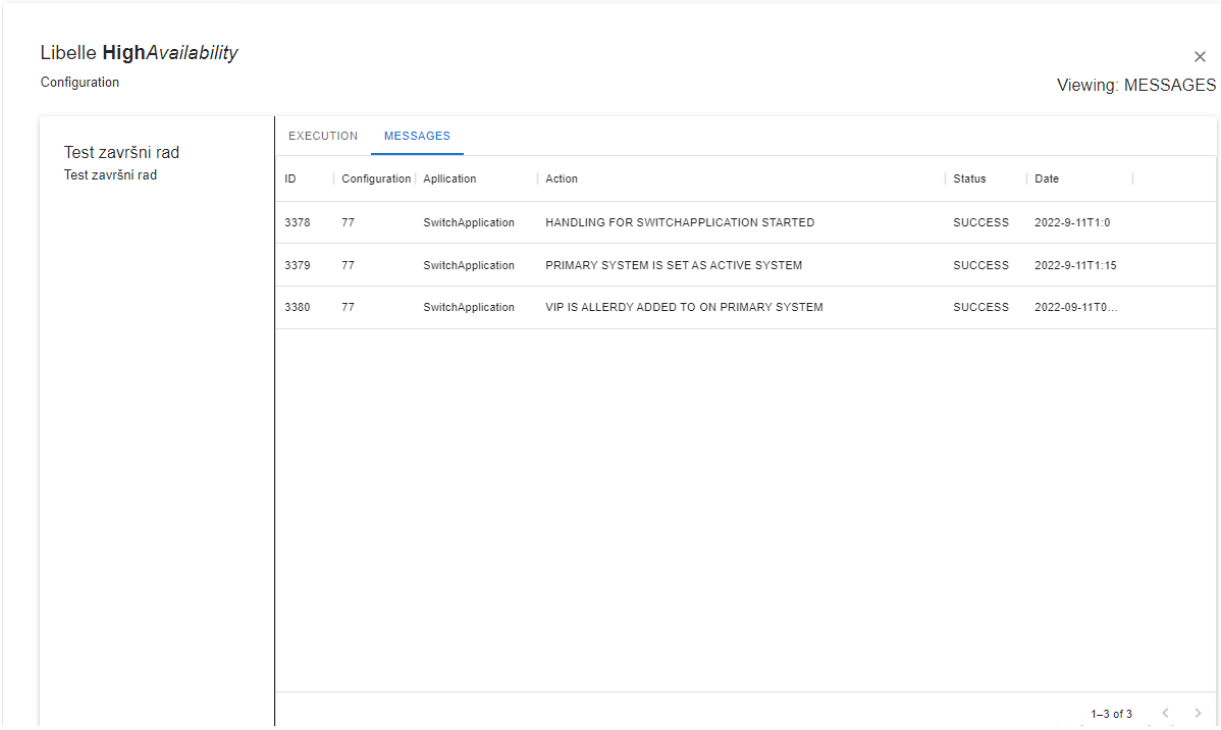


Slika 23. Dijalog za potvrđivanje promjene aktivnog sustava

Izvor: autor

Praćenje statusa izvedenih akcija moguće je klikom na polje „Messages“. Kad je korisnik navigiran u to polje, poruke o izvršavanju procesa osvježavaju se svakih 15 sekundi. U porukama

se mogu naći poruke za izvršeni zadatak iako je prije tih poruka ispisano zaustavljanje procesa; to se događa zato što se jednom pokrenuti proces mora dovršiti.



Slika 24. Praćenje statusa izvedenih akcija

Izvor: autor

8. ANALIZA FUNKCIONALNOSTI PROGRAMA

U postupku analize funkcionalnosti programa utvrdit će se nedostaci i vrijeme izvođenja implementiranog programa.

8.1 Nedostaci implementiranog programa

Program se sa svojim potrebama ne uklapa u dosadašnju arhitekturu proizvoda Libelle DataSuite jer je jedini proizvod koji je dosad implementiran u Libelle DataSuite imao drugačije potrebe, koje su bile izvršavanje zadatka na akciju koja je bila pokrenuta od strane aktera koji je upravljao proizvodom. U slučaju ponovnog pokretanja programa pokrenuta akcija ne bi se ponovo izvršavala te isti pristup ne zadovoljava potrebe implementiranog programa.

Potrebe implementiranog programa bile bi zadovoljene ako bi jednom pokrenuta akcija od strane aktera koji bi upravljao proizvodom bila permanentno pokrenuta akcija do trenutka zaustavljanja od strane aktera, neovisno o tome koliko puta bi Libelle DataSuite bio ponovno pokrenut.
Potencijalno rješenje za nedostatak programa bilo bi uvođenje takozvanog periodičkog izvođenja zadatka. Za periodičko izvođenje taskova može se koristiti Djanago Cellery.
Implementirani program nije dovoljno testiran kao ni sami Libelle DataSuite, radnje koje su izvođene na frontendu programa ne moraju biti stvarno pokrenute ili zaustavljene, već ovise o skripti koja se pokreće ručno. Tek nakon što je skripta pokrenuta radnje u frontendu programa odgovaraju stvarnim radnjama.

8.2 Vrijeme izvođenja implementiranog programa

Implementirani program kod izvođenja glavne metode algoritma programa u najgorem slučaju treba 83.63 sekundi za jednu konfiguraciju iako vrijeme varira, što je apsurdno vrijeme za izvođenje tako jednostavne akcije kao što je dodavanje i uklanjanje IP adrese ili dohvaćanje lista s podacima o mrežnim sučeljima.
Postavlja se glavno pitanje gdje je nastao problem. Sama srž problema nalazi se na *libellemaster* komponenti koja ima klasu za izvršavanje zadataka koji su poslani s master metoda, no ta je klasa predviđena na izvršavanje zadataka koji sadrže više od jednog zadatka, neovisno o tome je li se u zadatku nalazi jedan zadatak ili više zadataka. Tako se potroši previše vremena za slanje zadataka na *workere*.

9. ZAKLJUČAK

Pomoću programskog jezika Python i *web frameworka* Django, JavaScript biblioteke React i Material UI biblioteke implementiran je prototip programa za održavanje visoke dostupnosti servera baze podataka u trenutku neželjenih događaja. Odabirom Pythona za implementaciju logike programa vrijeme implementacije uvelike se smanjuje. Rok za izradu prototipa bio je četiri mjeseca, tri mjeseca bila su utrošena na samo planiranje programa, dok je jedan mjesec utrošen na implementaciju. U zadnjem koraku planiranja master metoda naišlo se na dva problema. Prvi je problem nemogućnost izvođenja periodičkih zadataka na *libellemaster* komponenti, dok se drugi problem javlja u sastavnom dijelu master komponente koja je također konceptualna te je stvorena za slanje zadataka na izvršavanje i spremanje rezultata zadataka. Orijentirana je na zadatke s dugim vremenom izvršavanja pa se puno vremena potroši na samu inicijalizaciju zadatka. Ova dva problema moraju se riješiti u budućnosti, implementirani program je spor i podložan mogućem zastoju (nema mogućnost implementirati *timeout*). Za daljnji razvoj programa predviđen je rok od godinu i pol tijekom kojeg će se korigirati svi dosadašnji propusti. Cjelokupni proces razvoja i istraživanja za kompaniju je pridonio obogaćivanju znanja i iskustva kod planiranja i implementacije programa.

10. POPIS LITERATURE

[1] https://www.educba.com/django-architecture/ (11.9.2022.)
[2] https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582 (11.9.2022.)
[3] https://reactjs.org/tutorial/tutorial.html (11.9.2022.)
[4] https://www.tutorialspoint.com/nodejs/nodejs_introduction.html (11.9.2022.)
[5] https://mui.com/material-ui/getting-started/overview/ (11.9.2022.)
[6] https://code.visualstudio.com/docs/ (11.9.2022.)
[7] https://www.pgadmin.org/docs/pgadmin4/latest/index.html (11.9.2022.)
[8] https://www.pgadmin.org/docs/pgadmin4/6.8/erd_tool.html (11.9.2022.)
[9] https://www.oracle.com/database/what-is-database/ (11.9.2022.)
[10] https://techdifferences.com/difference-between-mirroring-and-replication.html (11.9.2022.)
[11] https://www.enterprisedb.com/blog/what-does-database-high-availability-really-mean (11.9.2022.)
[12] https://docs.python.org/3/library/subprocess.html (11.9.2022.)
[13] https://docs.python.org/3/library/json.html#rfc-errata (11.9.2022.)
[14] https://www.javatpoint.com/dbms-er-model-concept (11.9.2022.)

11. PRILOZI

Popis slika

Popis kodova