Univerzitet u Novom Sadu
Fakultet tehničkih nauka u Novom Sadu

Mile Miljanović

Rukovanje pravilima u poslovnim veb aplikacijama

DIPLOMSKI RAD

- Osnovne akademske studije -

Novi Sad, 2020

	УНИВЕРЗИТЕТ У НОВОМ САДУ • **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6
	КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редниброј, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документације, **ТД:**	Монографска документација	
Тип записа, **ТЗ**:	Текстуални штампани материјал	
Врста рада, **ВР**:	Завршни (Bachelor) рад	
Аутор, **АУ**:	Миле Миљановић	
Ментор, **МН**:	Др Милан Сегединац, ванредни проф.	
Наслов рада, **НР**:	Руковање правилима у пословним веб апликацијама	
Језик публикације: **ЈП**:	Српски / латиница	
Језик извода, **ЈИ**:	Српски	
Земља публиковања, **ЗП**:	Република Србија	
Уже географско подручје, **УГП**:	Војводина	
Година, **ГО**:	2020	
Издавач, **ИЗ**:	Ауторски репринт	
Место и адреса, **МА**:	Нови Сад, Трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)	6/48/0/0/33/0/0	
Научна област: **НО**:	Електротехника и рачунарство	
Научна дисциплина, **НД**:	Рачунарске науке и информатика	
Предметна одредница / кључне речи, **ПО**:	Пословне веб апликације, пословна правила	
УДК		
Чува се, **ЧУ**:	У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:		
Извод, **ИЗ**:	У оквиру рада имплементирана је апликација за мобилне студенте, са циљем да се демонстрира рад правила у оквиру пословних веб апликација	
Датум прихватања теме, **ДП**:	16.06.2020	
Датум одбране, **ДО**:	02.09.2020	
Чланови комисије, **КО**:		
Председник	Др Горан Савић, ванредни проф.	
Члан	Др Милан Видаковић, редовни проф.	
Ментор	Др Милан Сегединац, ванредни проф.	Потпис ментора

UNIVERSITY OF NOVI SAD • **FACULTY OF TECHNICAL SCIENCES**

21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:		
Identification number, **INO**:		
Document type, **DT**:	Monographic publication	
Type of record, **TR**:	Textual printed material	
Contents code, **CC**:	Bachelor thesis	
Author, **AU**:	Mile Miljanović	
Mentor, **MN**:	Milan Segedinac, PhD, assoc. prof.	
Title, **TI**:	Handling the business rules in enterprise web applications	
Language of text, **LT**:	Serbian	
Language of abstract, **LA**:	Serbian	
Country of publication, **CP**:	Republic of Serbia	
Locality of publication, **LP**:	Vojvodina	
Publication year, **PY**:	2020	
Publisher, **PB**:	Author's reprint	
Publication place, **PP**:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	6/48/0/0/33/0/0	
Scientific field, **SF**:	Electrical engineering	
Scientific discipline, **ND**:	Computer science and informatics	
Subject / Keywords, **S/KW**:	Enterprise web applications, business rules	
UDC		
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:		
Abstract, **AB**:	This paper describes an implementation of an application for mobile students, with a goal of demonstrating the usage of business rules in enterprise web applications	
Accepted by sci. board on, **ASB**:	June 16th, 2020	
Defended on, **DE**:	September 2nd, 2020	
Defense board, **DB**:		
President	Goran Savic, PhD, assoc. prof.	
Member	Milan Vidakovic, PhD, full prof.	
Mentor	Milan Segedinac, PhD, assoc. prof.	Mentor's signature

Sadržaj

Spisak slika

Spisak skraćenica

SID – Synthesis of Integral Design
POJO – Plain Old Java Object
DDoS – Distributed Denial of Service
SSD – Solid State Drive
WAF – Web Application Firewall
SQL – Structured Query Language
HTTP – Hyper Text Transfer Protocol
REST – Representational State Transfer
JSON – JavaScript Object Notation
PDF – Portable Document Format
URI – Uniform Resource Identifier
UUID – Universally Unique Identifier
JPA – Java Persistence API
ORM – Object relational model

1 Uvod

1.1 Ciljevi rada

Od samih začetaka računarstva, postojala je ambicija da se automatizuje što je više moguće kompleksnih zadataka koje su izvršavali ljudi, odnosno eksperti u određenoj oblasti. Ova ambicija dovodi do rađanja ekspertskih sistema – kompjuterskih sistema koji simuliraju sposobnost donošenja odluka ljudskog eksperta, odnosno rešavanja domenski specifičnih problema koji zahtevaju rezonovanje nad nekim znanjem. Najčešće se ovakvi sistemi oslanjaju na određena pravila koja koriste da bi dolazili do relevantnih zaključaka. Ta pravila su obično u formatu AKO <uslov> ONDA <akcija>. U ovom radu je razmotren istorijat ovakvih sistema, kao i njihova struktura, način rada i navedeni su konkretni primeri ovakvih sistema. Posebno detaljan osvrt je na alat Drools, koji predstavlja kolekciju alata koji nam omogućavaju da rezonujemo nad logikom i podacima u poslovnim procesima. Ovaj alat je korišćen za implementaciju pravila u aplikaciji za studiranje u inostranstvu, koja treba da demonstrira rad sistema baziranih na znanju i koja je implementirana specifično za tu svrhu. Takođe, razmotrene su prednosti i mane sistema baziranih na znanju, i date su sugestije u kojim slučajevima ih je pogodno koristiti.

Studentski portal za mobilne studente je urađen kao web aplikacija koja se oslanja na pravila za zaključivanje validnosti studentskog formulara za studiranje u inostranstvu. U radu je detaljno razmotrena arhitektura aplikacije, struktura komponenti, i dodatno su pokriveni principi razvoja ovakvih aplikacija, obezbeđenje i struktura. Posebno poglavlje je namenjeno implementaciji aplikacije, gde možemo videti evaluaciju tehnologija korišćenih za razvoj, kao i slučajeve korišćenja od strane svih korisnika aplikacije, propraćene dijagramima i detaljima implementacije prestavljenim kroz kod.

1.2 Struktura rada

Ovaj rad se sastoji iz 5 poglavlja: uvoda, teorijskog pregleda sistema baziranih na znanju i poslovnih aplikacija u kom su pokriveni istorijat i teorijska osnova ekspertskih sistema i sistema baziranih na znanju, pregled tehnologija za pisanje pravila i teorijsko-praktični pregled Drools alata. Uključuje i teorijski osvrt na poslovne web aplikacije, njihovu strukturu, arhitekturu, principe razvoja i dizajna, kao i obezbeđenje. Sledeće poglavlje je arhitektura sistema, gde je dat detaljan prikaz svih komponenti i entiteta u sistemu, kako sa serverske tako i sa klijentske strane, zatim implementacija, gde su pružene informacije o odabranim tehnologijama za implementaciju, pokriveni slučajevi korišćenja od strane svih korisnika aplikacije i nudi uvid u kod, tj. konkretnu implementaciju, i na kraju zaključak – završna reč, sumira rezultate rada i daje smernice za dalje istraživanje.

2 Sistemi bazirani na pravilima i poslovne veb aplikacije

2.1 O sistemima baziranim na pravilima i ekspertskim sistemima

U računarstvu, sistemi bazirani na pravilima služe za skladištenje i manipulaciju znanja, kako bi se interpretirale informacije na način koristan za problem koji aplikacija rešava. Često se koristi u aplikacijama koje se oslanjaju na računarsku inteligenciju, kao i u istraživanju računarske inteligencije. Termin *sistem baziran na znanju* se uglavnom koristi za sisteme koji uključuju pravila definisana od strane ljudi koji se smatraju ekspertima u domenu problema. Klasičan primer sistema baziranog na znanju je domenski specifičan *ekspertski sistem* koji koristi pravila da dođe do zaključaka ili izbora. Primera radi, ovakvi sistemi mogu da pomognu doktorima da dođu do tačne dijagnoze, na osnovu unesenih simptoma, ili da se odaberu taktički najoptimalniji potezi u nekoj igri, na primer šahu. Ovakvi sistemi se takođe mogu koristiti u leksičoj analizi, ili u procesiranju prirodnog jezika.

Pod ekspertskim sistemima se podrazumevaju sistemi bazirani na znanju, koji pri rezonovanju simuliraju sposobnost donošenja odluka karakterističnih za ljudskog eksperta. Oni su dizajnirani da rešavaju kompleksne probleme rezonovanjem kroz bazu znanja uz pomoć pravila, koja su uglavnom predstavljena kroz if-then strukturu. Ekspertski sistemi se sastoje iz dva podsistema: baze znanja i mehanizma za zaključivanje (rezonera). Baza znanja obuhvata pravila i činjenice (fakte), odnosno, adekvatno reprezentovano znanje relevantno za domen problema, a mehanizam za zaključivanje primenjuje pravila na postojeće činjenice, da bi došao do zaključaka, i iz njih izvukao nove činjenice.

Sistemi bazirani na pravilima se prvi put pojavljuju 1965. godine na univerzitetu Stanford, gde su istraživači, predvođeni Edvardom Fajgenbaumom (slika 1), koji se smatra ocem ekspertskih sistema, pokušali da stvore inteligentan sistem opšte svrhe, koji bi bio sposaban da rešava probleme svakog tipa. Očekivanja su bila visoka, ali je istraživanje završeno neuspehom, jer je ovaj pokušaj bio previše ambiciozan. Kasnije su shvatili da bi bilo bolje da se ograniče na jedan specifičan domen problema, pa su krajem 60ih godina pokušali da identifikuju domene u kojima je ekspertsko znanje visoko cenjeno i kompleksno, kao što je dijagnostika bolesti (Mycin, 1972), identifikacija nepoznatih organskih molekula (Dendral, 1969), geološka analiza tla i minerala (Prospector, 1979), ili rešavanje matematičkih problema (MACSYMA, 1982). Ovo dovodi do prvih velikih uspeha računarske inteligencije.

Prvi sistem baziran na pravilima koji je imao kapacitet da bude veliki komercijalni uspeh je bio SID (Synthesis of Integral Design), razvijen 1982. godine. Pisan u LISP-u, SID je generisao 93% logičkih kola za superračunar VAX-9000. Ulazni parametri su bila pravila koja su ručno pisana od strane eksperata, a SID je proširivao pravila i generisao rutine za logičku sintezu, koje su bile višestruko veće od samih izvornih pravila. Kombinacija generisanih pravila je rezultovala dizajnom koji je nadmašio sposobnosti samih eksperata, i u većini slučajeva bolje funkcionisala od pravila pisanih ručno. Nakon završetka VAX-9000 superračunara, projekat SID je završen i nije više nikad korišćen.

Moderni sistemi bazirani na pravilima i dalje primenjuju ideje ustanovljene od strane pionira, ali su često integrisani unutar velikih, kompleksnih sistemima. Ograničenja na koja su rani istraživači nailazili su ih naterala da potraže nove metode pristupa problemu, da bi se što vernije replicirao ljudski proces donošenja odluka. Često su ti pristupi bazirani na novijim metodama u računarskoj inteligenciji, kao što je *mašinsko učenje* i *data mining*. Moderni sistemi efikasnije pripajaju novo znanje i samim tim se efikasnije ažuriraju. Mogu da izvode generalizacije iz postojeće baze znanje i da obrađuju velike količine

kompleksnih podataka – iz čega nastaje *big data* koncept. Ponekad se ovakvi sistemi nazivaju "inteligentni sistemi".

Važno je napomenuti da sistemi bazirani na pravilima NISU veštačka inteligencija. Bitna karakteristika veštačke inteligencije jeste da je ona u stanju da sama odluči koju akciju da preduzme; može da uči i da se adaptira. Sistemi bazirani na pravilima samo izvršavaju pravila tačno onako kako su definisana od strane ljudi. Dakle, sistem ne radi za sebe i ne pravi "inteligentne" odluke, niti se sam ažurira i ne uči iz svojih grešaka.

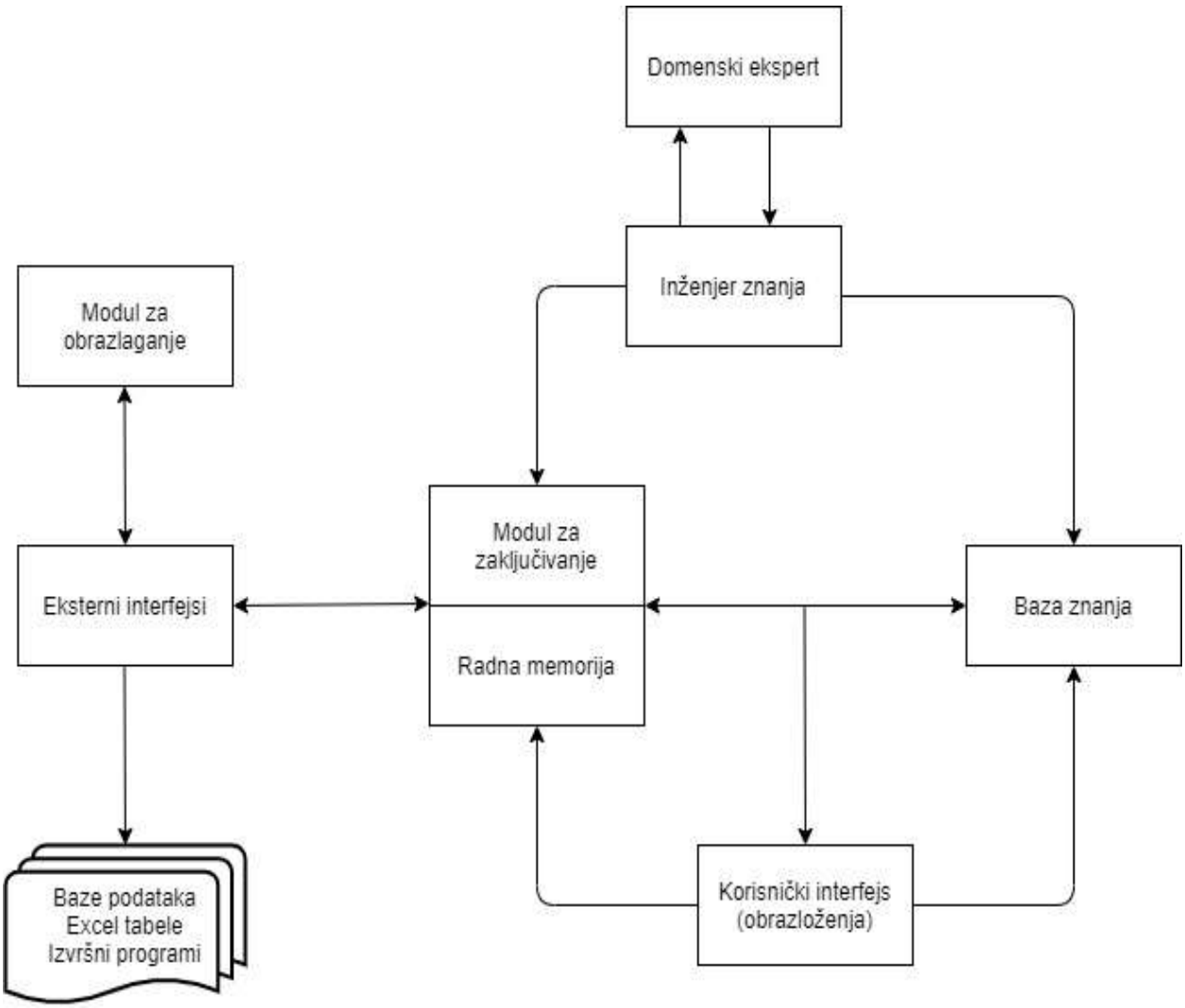
2.2 Struktura sistema baziranih na pravilima

Tipičan sistem baziran na pravilima sadrži 4 komponente, prikazane na slici 1: listu pravila, ili bazu pravila, koja predstavlja specifičnu vrstu baze znanja. Pravila su obično u formatu IF-THEN: postoje uslov (tzv. leva strana pravila) i akcija (tzv. desna strana pravila). Ako je uslov zadovoljen, akcija se izvršava. Cilj je da se automatizuju procesi, uz razbijanje tih istih procesa na korake.

Sledeća komponenta je modul za zaključivanje (rezoner), koji vrši zaključivanje na osnovu ulaznih podataka i baze pravila. Zaključivanje se vrši u 3 koraka: podudaranje, odnosno provera da li su uslovi pravila zadovoljeni, nakon čega se kreira uređena lista elemenata radne memorije koji zadovoljavaju levu stranu pravila, zatim razrešenje konflikta, odnosno određivanje redosleda u kom će se desne strane pravila koja zadovoljavaju uslove izvršiti i akcija, odnosno izvršenje desnih strana pravila u redosledu određenom tokom faze razrešenja konflikta.

Svaki sistem baziran na pravilima takođe sadrži modul za obrazlaganje, koji objašnjava rezonovanje sistema krajnjem korisniku. Kredibilitet sistema se uspostavlja onog momenta kada može da objasni *kako i zašto* je došao do nekog zaključka. Da bi obrazložio kako je došao do zaključka, modul za obrazlaganje prati lanac pravila koja su izvršena tokom interakcije sa korisnikom, i deskriptivno objašnjava kako je dedukovao određenu činjenicu i zašto jeste ili nije koristio određena pravila. Modul za obrazlaganje takođe mora da objasni zašto je određena informacija potrebna da bi se kompletirao korak u rezonovanju.

Preostale komponente su privremena radna memorija, koja sadrži činjenice (fakte), i na kraju, korisnički interfejs ili neku drugu vrstu povezivanja sa spoljnim svetom, koja služi za slanje i primanje ulaznih/izlaznih signala i komunikaciju sa korisnikom.



Slika 1 - Struktura sistema baziranog na pravilima

2.3 Inženjerstvo znanja

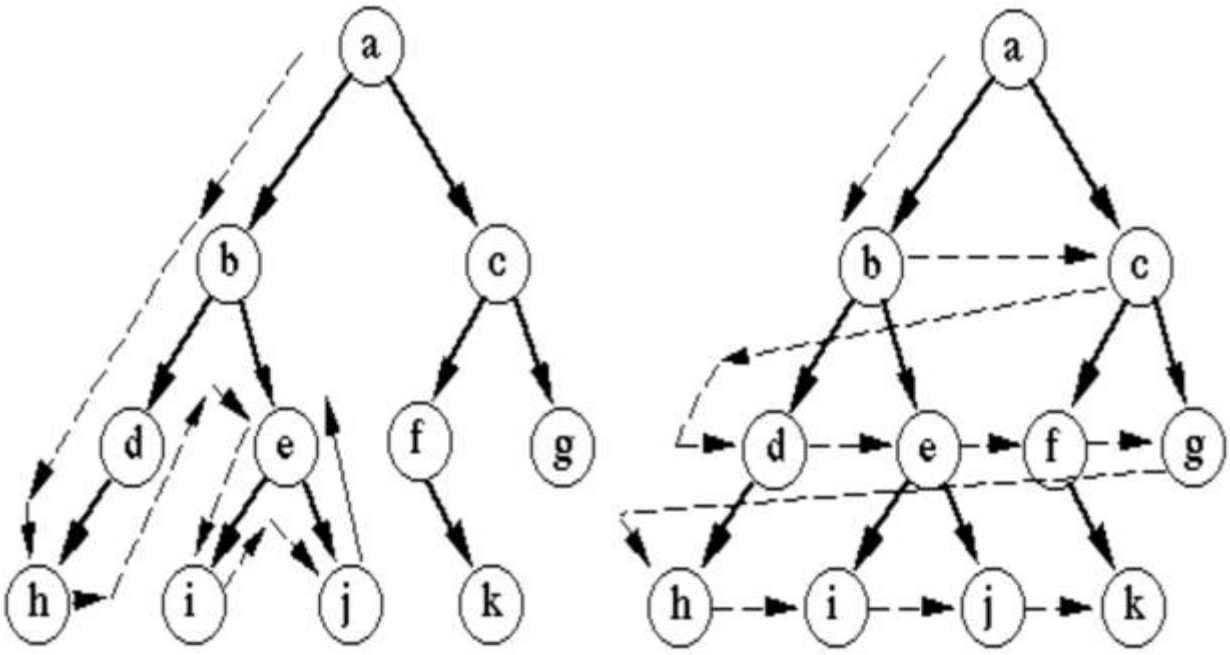
Bitan proces u razvoju sistema baziranih na pravilima je inženjerstvo znanja. Ključni ljudi koji su uključeni u razvijanje sistema su inženjer znanja, ekspert za domen i krajnji korisnik. Nakon što inženjer znanja prikupi dovoljno generalnih informacija o domenu problema, on treba da se konsultuje sa domenskim ekspertom kako bi se rešili postojeći problemi oko dizajna, i nakon toga sistem je spreman za razvijanje – bira se način reprezentacije znanja, dizajn korisničkog interfejsa, itd. Po završetku dizajna, inženjer znanja pravi prototip sistema, koji treba da bude u stanju da reši probleme u malom poddomenu problema. Nakon što je prototip završen, inženjer znanja i domenski ekspert testiraju i dorađuju znanje davajući sistemu sve više problema iz domena koje treba da reši, i usput koriguju nedostatke.

2.4 Strategije zaključivanja

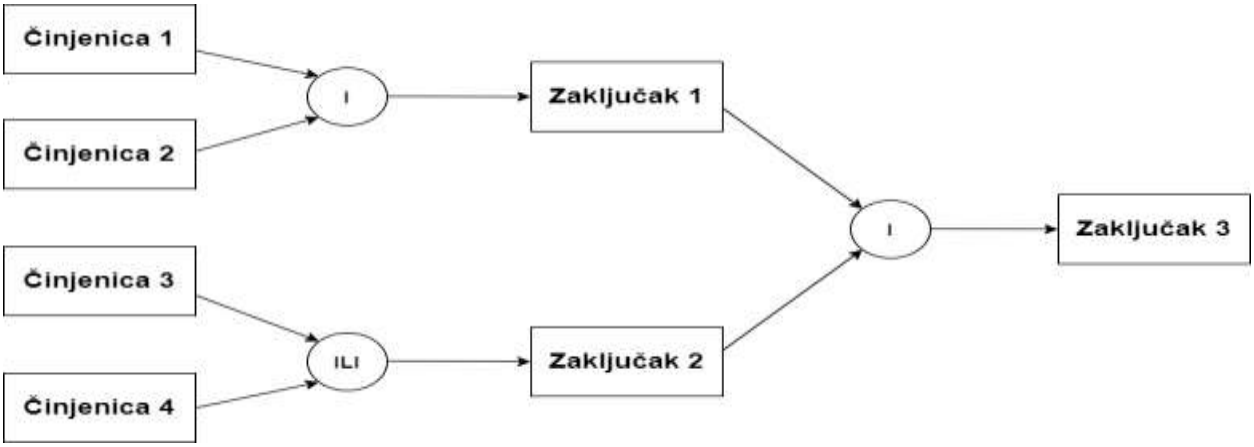
Modul za zaključivanje može da vrši zaključivanje putem 2 strategije: ulančavanje unapred i ulančavanje unazad.

Kod ulančavanja unapred, počinje se od poznatih činjenica, i pomera se unapred primenom pravila na te činjenice, kako bismo došli do novih činjenica, i tako sve dok ne stignemo do cilja (zaključka), što vidimo na slici 3. Zbog ovoga se ova strategija naziva i *zaključivanje vođeno podacima (data-driven),* ili *pristup odozdo na gore*. Izvršavanjem desnih strana pravila za one fakte čije su leve strane zadovoljene, dolazimo do zaključaka koje dalje možemo koristiti da bismo došli do novih zaključaka. Prolazi se kroz sva moguća pravila i koristi se strategija pretrage poznata kao *prvi u širinu* (slika 2). Pretraga prvi u širinu – pretraga počinje od korenskog elementa, i pretražuje sve susedne elemente na grafu, odnosno elemente koji su na istom nivou dubine, pre nego što krene na sledeći nivo. Koristi se za zadatke kao što su planiranje, dijagnostika, klasifikacija, monitoring dizajn procesa...

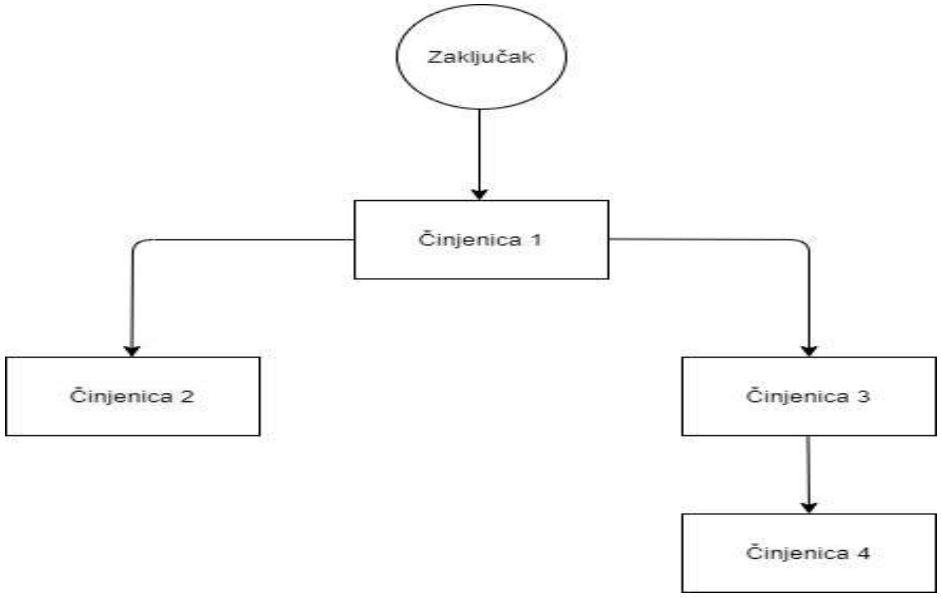
Kod ulančavanja unazad, počinje se od cilja (zaključka) i ide se u nazad, odnosno izvršavaju se pravila kako bi se dobile činjenice koje podržavaju zaključak, kao što je prikazano na slici 4. Zbog ovoga se ova strategija naziva i *zaključivanje vođeno ciljem (goal-driven),* ili *pristup odozgo na dole.* Iz cilja se izvlače podciljevi dok se ne pronađu činjenice koje podržavaju sve podciljeve, a samim tim i cilj. Ne prolazi se kroz sva pravila, nego samo kroz ona neophodna da se dobiju zadovoljavajuće činjenice i koristi se strategija pretrage poznata kao *prvi u dubinu* (slika 2). Pretraga prvi u dubinu – pretraga počinje od korenskog elementa, pomera se dalje za jedan nivo i ide što dublje može pre nego što se vrati za jedan nivo unazad i pretraži susedni čvor. Koristi se za zadatke kao što su dijagnostika, preskripcija, debagovanje aplikacija...



Slika 2 - Pretraga prvi u dubinu (levo) i prvi u širinu (desno)



Slika 3 - Ulančavanje unapred



Slika 4- Ulančavanje unazad

2.5 Programski jezici bazirani na pravilima

U ovom poglavlju biće navedeni neki primeri programskih jezika baziranih na pravilima.

AWK je jezik za procesuiranje teksta, ekstraksciju podataka i alat za izveštavanje. CLIPS predstavlja najrasprostranjeniji alat za ekspertske sisteme, objektno orijentisane prirode. Constraint Handling Rules je deklarativni jezik namenjen rešavanju kombinatornih problema, takođe ima primenu u gramatičkoj indukciji, abduktivnom rezonovanju, multiagentskim sistemima, procesuiranju prirodnog jezika, prostorno-vremenskom rezonovanju, itd. Zatim imamo Drools, koji ćemo detaljno razmotriti u okviru

ovog rada. GOAL agentski programski jezik, koji se koristi za programiranje kognitivnih agenata. Jess je deklarativni jezik namenjen za Java platformu, nadskup CLIPS-a, i ima primenu u automatizaciji ekspertskih sistema. OPS5 je prvi jezik korišćen u uspešnom ekspertskom sistemu, za konfigurisanje VAX računara. Prolog je logički programski jezik, sa korenima u logici prvog reda, danas jedan od najpopularnijih jezika ovog tipa. Ima široku primenu u automatizaciji planiranja, ekspertskim sistemima, dokazivanju teorema, procesuiranju prirodnog jezika, itd. ToonTalk je programski sistem namenjen za decu, prezentovan u formi animiranih likova. Mathematica je široko rasprostranjen moderan računarski sistem koji podržava neuronske mreže, mašinsko učenje, procesuiranje slika, geometriju, vizualizacije, itd. Wolfram je jezik koji podržava velik broj paradigmi, sa akcentom na funkcionalnom programiranju, simboličkom računanju i programiranje bazirano na pravilima.

2.6 Prednosti i mane sistema baziranih na pravilima

Korišćenje sistema baziranih na pravilima nosi sa sobom niz prednosti. U nastavku će biti navedene neke od njih:

Pravila se izražavaju deklarativno – omogućuju nam da kažemo šta želimo da se uradi, a ne kako da se uradi. Dakle, pravila su mnogo čitljivija od koda i često se rešenja težih problema mogu lakše predstaviti pravilima. Zatim, postoji koncept razdvajanja poslovne logike od ostatka koda – poslovna logika je često podložna promenama, i ako je izražena u pravilima, lakša je za menjanje i održavanje. Na ovaj način, sva poslovna logika se nalazi na jednom mestu. Sistemi bazirani na pravilima često su praćeni algoritmima koji efikasno uparuju podatke i pravila (npr Rete algoritam, Leaps algoritam, itd.), i na taj način omogućuju brzinu i skalabilnost. Takođe, primenjuju centralizaciju znanja – koristići pravila kreira se repozitorijum znanja koji služi kao izvor istinitosti za poslovnu logiku – idealno su pravila toliko čitljiva da mogu da služe i kao dokumentacija. Dobri sistemi bazirani na pravilima obično imaju sposobnost objašnjavanja, odnosno, pravila trebaju da nas obaveštavaju o svim odlukama, kao i zašto su one donete. Još jedna bitna karakteristika je razumljivost – sintaksa pravila je bliža prirodnom jeziku, i samim tim je razumljivija ljudima koji nemaju tehničke sposobnosti programera; i postoji mogućnost da se menjaju pravila bez ponovnog startovanja aplikacije

Neki od nedostataka sistema baziranih na pravilima: alati koji barataju pravilima su često resursno zahtevni i mogu značajno da utiču na performans. Takođe postoji problem narušavanja objektno orijentisanog koncepta na kom počivaju mnoge aplikacije, jer se logika prenosi u pravila umesto da ostaje unutar objekata. Sa programerske strane, može biti teško testirati i refaktorisati deklarativno definisana pravila i integrisati ih sa kodom koji je pisan na drugačiji način, a sa povećanjem broja pravila, postaje sve teže održavati ih – mora se voditi računa o tome koja pravila su međusobno zavisna i koje objekte menjaju, o prioritetu izvršavanja pravila, itd. Postoji i opasnost od takozvane “eksplozije pravila“ – u pokušaju da se reši problem, često dolazi do prevelikog broja pravila koja se preklapaju i koja mogu biti kontradiktorna; sistem postaje kompleksan i nerazumljiv. I na kraju, sintaksa pravila se mora naučiti i to zahteva vreme – što programeri često nisu voljni da rade.

Sistemi bazirani na pravilima se uglavnom koriste kada ne postoji zadovoljavajuće tradicionalno programersko rešenje problema, ili kada je poslovna logika podložna čestim promenama. Takođe se koriste kada domenski eksperti nemaju adekvatno programersko znanje, jer ih je lakše podučiti sintaksi pravila nego sintaksi programskog jezika.

Sistemi bazirani na pravilima ne bi trebali da se koriste ukoliko je mali projekat, sa manje od 20 pravila, ili ukoliko je poslovna logika dobro definisana i statična, nepodložna promenama. Takođe, ukoliko su pravila previše prosta, i uključuju rad nad malim brojem objekata, nije potrebno koristiti ih. U slučaju da je primarni fokus na performansu, ili problem nije prikladan za pisanje u formi pravila, optimalnije je ostaviti logiku u samom kodu i ne koristiti pravila.

2.7 Programski jezik Drools

Drools je integraciona platforma za poslovnu logiku pisana u Javi, podržana od strane Jboss-a i Red Hat-a koja implementira Rete algoritam, o kom će više reči biti kasnije. Predstavlja kolekciju alata koji nam omogućavaju da rezonujemo nad logikom i podacima u poslovnim procesima. Drools možemo podeliti u 2 dela: autoring, odnosno proces pisanja pravila, i vreme izvršavanja (runtime), odnosno proces pravljenja objekata u radnoj memoriji i regulisanja aktivacije pravila. Kao i kod većine jezika baziranih na pravilima, pravila u Drools-u su u formatu if-then – ako je neki uslov zadovoljen, izvrši akciju.

```
rule  <rule_name>
    <attribute> <value>

    when
        <conditions>

    then
        <actions>
end
```

Slika 5 - Pseudokod jednog pravila

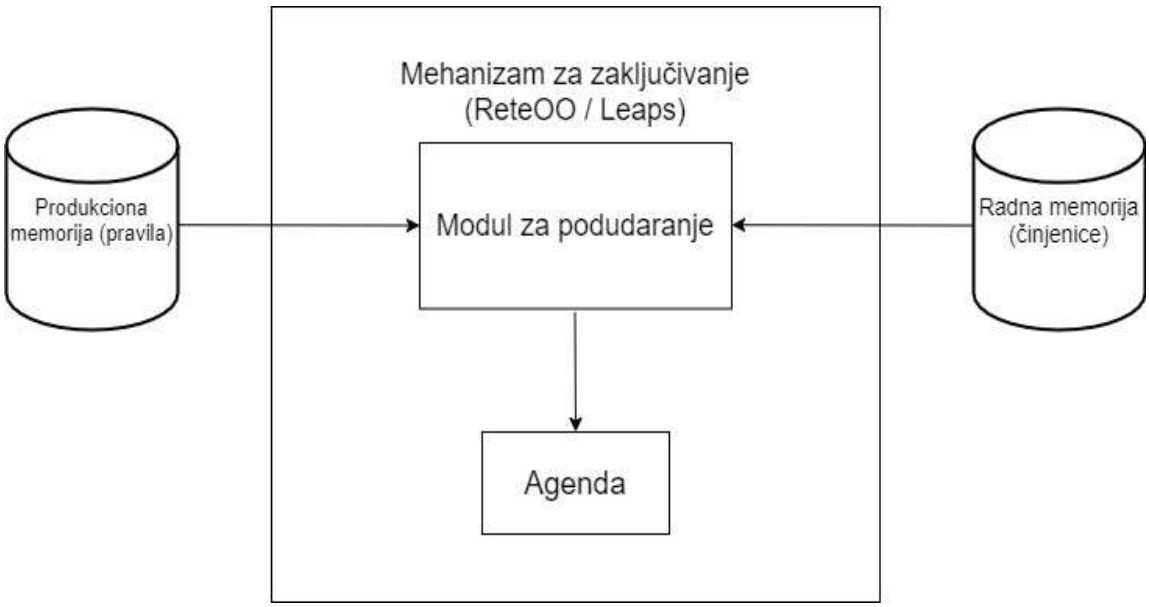
Proces uparivanja postojećih činjenica sa pravilima se naziva *podudaranje šablona* (*pattern matching*), i njega izvršava modul za zaključivanje. Algoritam za sprovođenje ovog procesa se u slučaju Drools-a zove ReteOO algoritam – poboljšana i optimizovana verzija Rete algoritma za objektno orijentisane sisteme. Drools radi po principu hibridnog ulančavanja – može da radi i ulančavanje unapred i unazad. Drools se lako integriše sa postojećim razvojnim okruženjima kao što su Eclipse, IntelliJ i Visual Studio Code, koja nude podršku za debagovanje, validaciju i pisanje pravila. Pseudokod pravila je prikazan na slici 5.

U nastavku su navedeni ključni termini koji se koriste u okviru Drools-a.

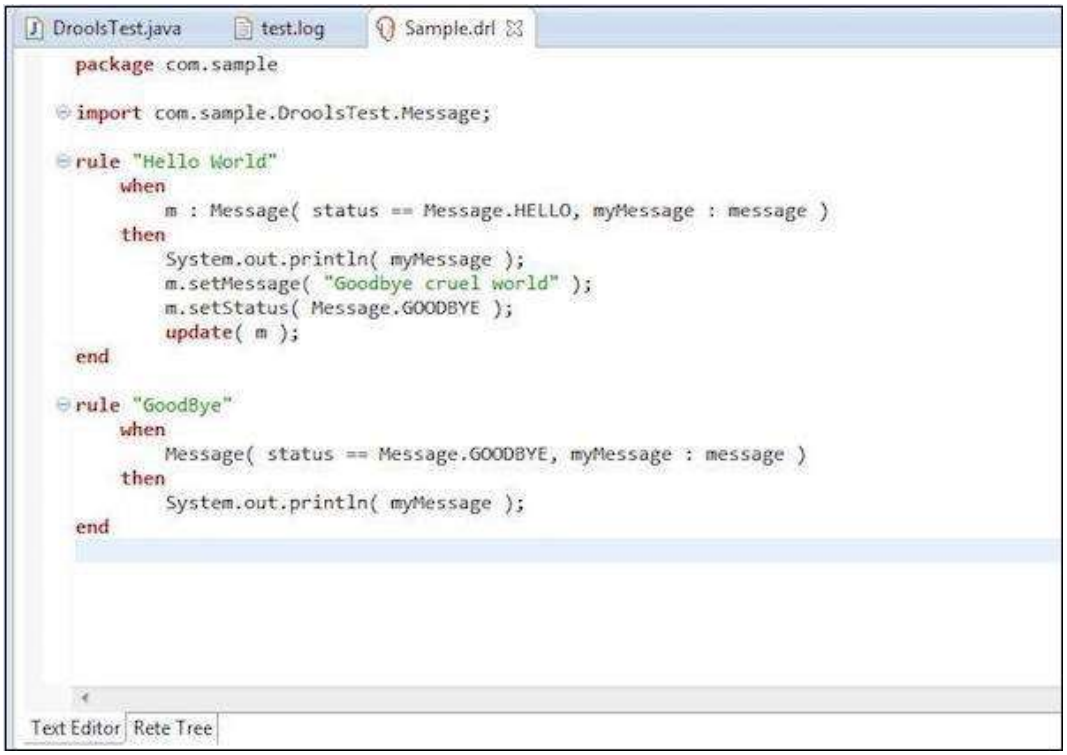
Pravila predstavljaju fajlovi sa ekstenzijom .drl u kojima navodimo uslove i akcije koje treba da se izvrše ako su uslovi zadovoljeni (when ‘a’ then ’b’). Činjenice su podaci nad kojima se pravila izvršavaju. U Javi, to su obično POJO objekti. Sesija je glavna komponenta koja služi za okidanje pravila. Činjenice se ubacuju u sesiju i kada su uslovi ispunjeni, izvršavaju se desne strane pravila. Postoje dva tipa sesije: stateless i stateful. U stateless sesiji se stanje sesije između poziva ne održava, za svaki poziv se pravi nova sesija, a stateful pamti stanje između poziva. Aktivacija predstavlja akciju, odnosno desnu

stranu pravila. Agenda je logički koncept, mesto na kom aktivacije čekaju da budu izvršene, odnosno mesto na kom se nalaze sve aktivacije čiji su uslovi zadovoljeni.

Strukturu Drools-a možemo videti na slici 6, a konkretan primer pravila na slici 7.



Slika 6 - Struktura Drools-a



Slika 7 - Primer jednog pravila u Drools-u

2.8 Rete algoritam

Ovaj algoritam je razvio američki naučnik Charles L. Forgy kasnih 70ih godina, ali je prošlo dosta vremena pre nego što je ovaj algoritam postao popularan. Pojavom tehnologije poslovnih pravila, algoritam dobija na popularnosti. Algoritam je prošao kroz dosta izmena od inicijalne verzije, do konačne verzije 1982. godine koju je Forgy predao kao doktorat. Naziv Rete potiče od latinske reči za mrežu – graf algoritma u velikoj meri zaista podseća na mrežu. Suština algoritma jeste da uklanja potrebu za suvišnim kalkulacijama – on pamti kako celokupna, tako i parcijalna zadovoljenja uslova.

Ideja koju je Forgy imao jeste da razdvoji evaluaciju hipoteze od redosleda izvršavanja. Razlog za to je situacija kada imamo veliki broj poslovnih pravila koja trebaju da se primene na činjenice, konstantna evaluacija i reevaluacija pravila i uređivanje redosleda izvršavanja može da bude resursno veoma zahtevno. Forgy-jev pristup zaključivanju omogućuje veliku uštedu na resursima – rezultuje sa manjim iskorišćenjem memorije i bržom evaluacijom pravila. Treba imati na umu da Rete algoritam nije optimalan u svim situacijama; on idealno radi kada imamo veliki broj pravila i manji broj objekata u radnoj memoriji. Za obrnutu situaciju, ukoliko je performans bitan, bilo bi dobro potražiti druga rešenja.

Tradicionalni pristup bi bio uređivanje i ugneždenje pravila tako da se izvršavaju sekvencijalno. Ovaj pristup nije primenljiv u slučaju kada pravilo treba da se izvrši kao rezultat izvršavanja drugih pravila. U tradicionalnom pristupu, ukoliko su pravila prioritizovana, sva pravila bi trebala ponovo da se evaluiraju nakon svakog izvršenja desne strane. Ukoliko pravila nisu prioritizovana, ponovna evaluacija bi se vršila nakon prolaska kroz sva pravila.

Primer pravila za korisnika avio kompanije:

Ako je ukupna kilometraža letova u prošloj ili tekućoj godini > 25000km onda je Status = Srebrni

Ako je ukupna kilometraža letova u prošloj ili tekućoj godini > 100000km onda je Status = Zlatni

Ako je kilometraža leta manja od 500km onda dodeli 500km na kilometražu leta

Ako je kilometraža leta veća od 500km onda dodeli tu vrednost na kilometražu leta

Ako je klasa biznis ili prva, dodeli bonus 50% na kilometražu leta

Ako je status Zlatni i nije partner let, dodeli bonus 100% na kilometražu leta

Ako je status Srebrni i nije partner let, dodeli bonus 20% na kilometražu leta

Ako je status Srebrni i u toku jednog meseca je imao više od 5 letova, dodeli bonus 1000km

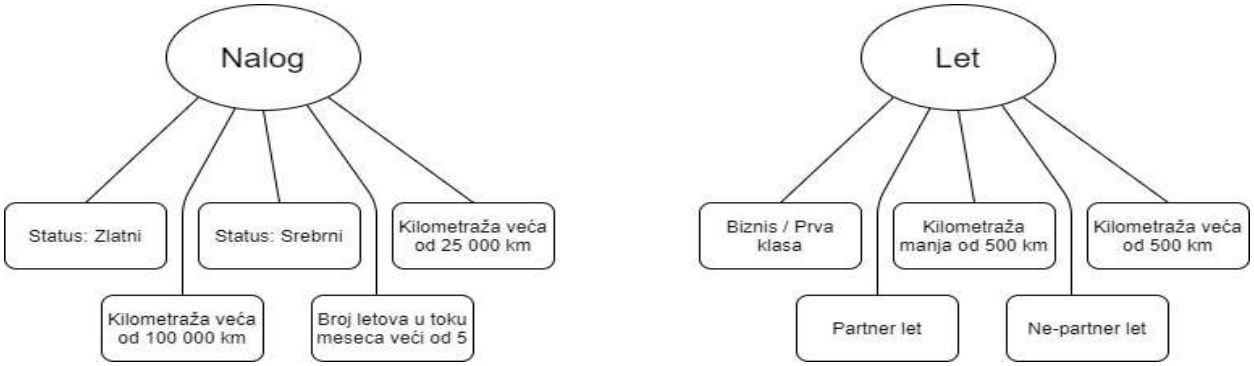
Ako je status Zlatni i u toku jednog meseca je imao više od 5 letova, dodeli bonus 3000km

Ako prođemo sekvencijalno kroz ova pravila, i korisnik dobije, recimo, zlatni status u toj transakciji, sva pravila moraju ponovo da se izvrše da bi taj korisnik dobio sve odgovarajuće bonuse.

Rete mreža predstavlja osnovu algoritma. Sačinjena je od čvorova koji sadrže listu objekata koji zadovoljavaju određeni uslov. Diskriminantno stablo je prvi deo Rete mreže. Počinje sa *alfa čvorovima* koji se asociraju sa klasama (u objektno-orijentisanom smislu). Sve instance jedne klase će biti elementi jednog alfa čvora. U slučaju primera avio kompanije, imamo alfa čvorove za Let i Nalog, na slici 8:

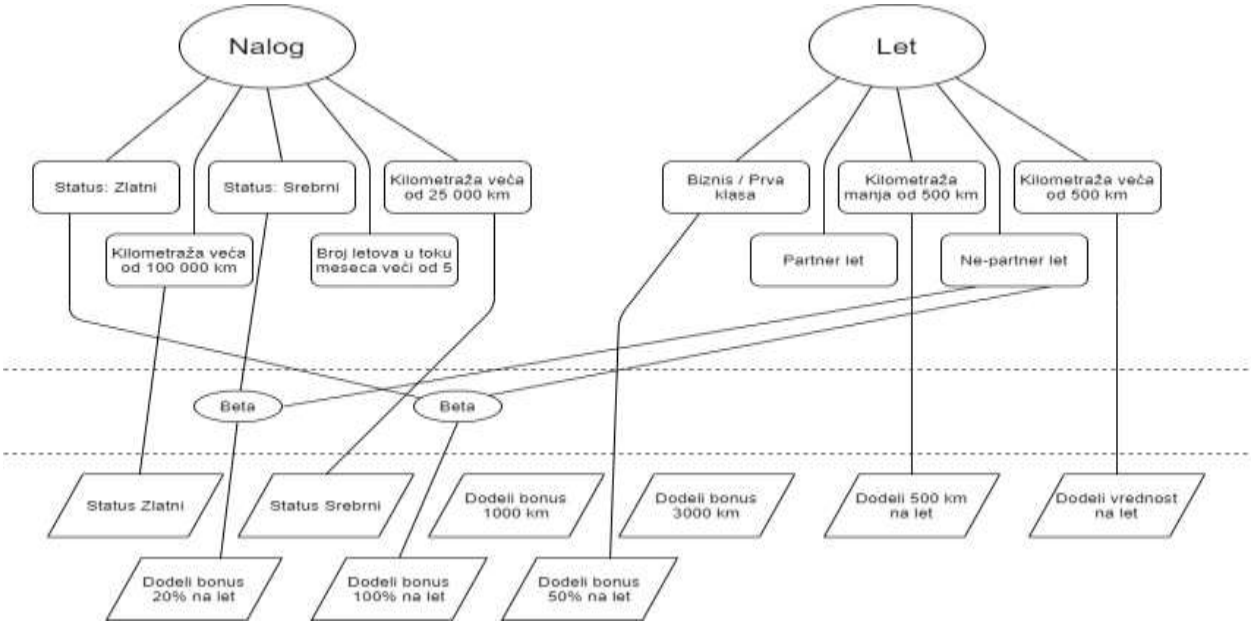
Slika 8 – Čvorovi za Nalog i Let

Diskriminacija se dešava nakon što dodamo uslove pravila. Svaki novi čvor predstavlja test niza uslova koji su definisani na višim nivoima. Ovi čvorovi formiraju tzv. *alfa mrežu*. Ako idemo odozgo na dole, možemo pročitati sve uslove koji važe za jedan tip objekata, prikazane na slici 9:



Slika 9 – Alfa mreža za Nalog i Let

Konačno, spajamo čvorove različitih klasa. Možemo da kombinujemo uslove, na primer možemo uzeti nepartnerski let od strane zlatnog člana. Ovo se nazivaju *beta čvorovi*, odnosno kombinacije liste objekata iz jedne grane koji zadovoljavaju određene uslove (tzv. levi ulaz) sa listama objekata iz druge grane koji takođe zadovoljavaju određene uslove (tzv. desni ulaz), i oni formiraju tzv. *beta mrežu*. Stablo se završava sa akcijom pravila, kao što se vidi na slici 10:



Slika 10 – Celokupan prikaz Rete mreže za Nalog i Let

Sa aspekta performansa, ne bi trebalo kombinovati previše šablona u jednom pravilu. Jasno je sa grafova, da bi produkt svih mogućih naloga i svih mogućih letova rezultovao sa kombinatornom eksplozijom. Znači, što više diskriminacije unapred, to bolje.

Bitno je napomenuti da su uslovi pravila povezani logičkim i (AND), a logičko ili uglavnom nije podržano. Ukoliko želimo da dodamo na primer, isti bonus za srebrnog i zlatnog člana, pravila se dupliciraju, odnosno moramo napisati jedno pravilo za zlatnog, i jedno za srebrnog člana. Sa aspekta performansa ovo nije problem, zato što je Rete algoritam optimizovan za rad sa velikim brojem pravila.

Faza evaluacije – puštanje podataka kroz Rete mrežu, da bi se identifikovala relevantna pravila čiji su uslovi zadovoljeni. Čvorovi mreže sadrže liste objekata koji zadovoljavaju uslove.

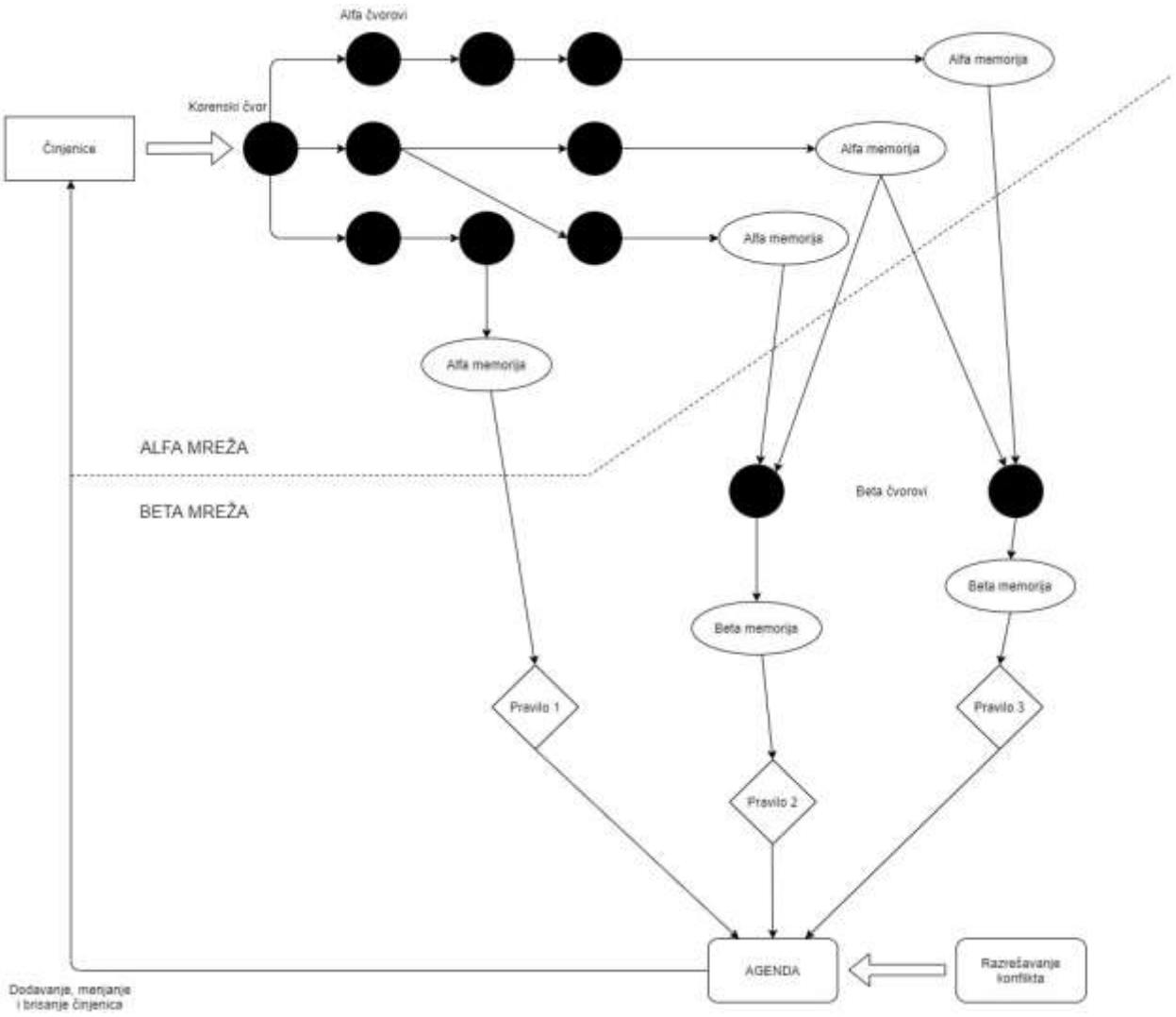
Faza izvršavanja – za pravila čiji su svi uslovi zadovoljeni kažemo da su aktivna u agendi. Agenda sadrži listu svih pravila koja trebaju da se izvrše, zajedno sa listom svih objekata koji su uticali na to da pravilo dospe u agendu. Agenda će sortirati pravila na osnovu prioriteta izvršavanja, ili nekog drugog mehanizma razrešenja konflikta. Ukoliko neko pravilo višeg prioriteta invalidira pravilo nižeg prioriteta, ono se naravno neće izvršiti.

Ponavljanje ciklusa – nakon izvršavanja prvog pravila, činjenice se ponovo propagiraju, i ukoliko je došlo do promene nad činjenicama, odgovarajuća pravila će se ponovo evaluirati, i to samo ona za koja su relevantna polja promenjena (na primeru avio kompanije, pravilo da li korisnik ima status gold/silver zavisi od ukupne kilometraže – samo ako je došlo do promene ukupne kilometraže, ova pravila će biti reevaluirana). Ona pravila koja su već prošla evaluaciju i čija relevantna polja nisu promenjena, ostaju u memoriji i neće se ponovo evaluirati.

Pun primer: Nikola leti iz Beograda za Stokholm – kilometraža leta je 2300 km. Nikola ima ukupnu kilometražu od 23400 km. Nikola ne ispunjava uslove za srebrnog ili zlatnog člana. Let ima više od 500km, pa ćemo mu dodeliti tu vrednost leta. Nikola leti ekonomskom klasom, pa nema bonuse za biznis/prvu klasu. Pošto nije zlatni ili srebrni član, ne dobija bonuse na let za zlatne/srebrne članove. Dodajemo 2300 km na postojećih 23400 km što rezultuje sa 25700 km. Promenjena je ukupna kilometraža, gledamo da li sad ispunjava uslov za zlatnog/srebrnog člana. Uslov za srebrnog je 25000 km, tako da Nikola sada postaje srebrni član. Pošto je i to promenjeno, moramo proveriti koji su bonusi za srebrnog člana. Srebrni članovi dobijaju 20% bonus kilometraže na trenutni let, tako da se dodaje 460 km, odnosno 20% od leta na Nikolinu ukupnu kilometražu.

Opšti grafički prikaz algoritma možemo videti na slici 11.

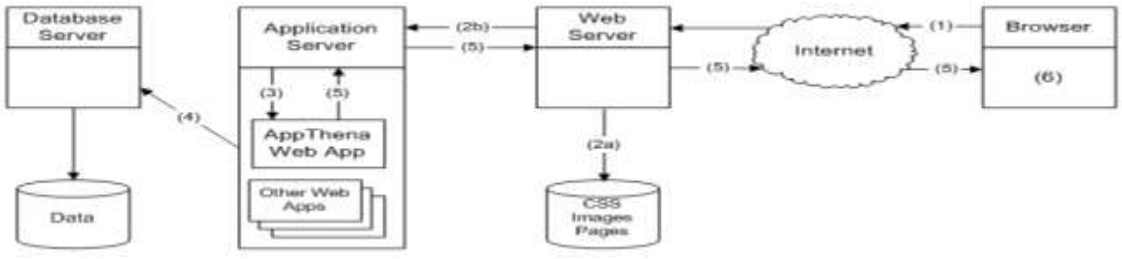
Postoje 3 varijacije Rete algoritma: Rete II – 1980ih, Charles Forgy je razvio naslednika Rete algoritma, nazvan Rete II. Detalji implementacije ovog algoritma, za razliku od prvobitne verzije, nisu otkriveni. Rete II ima višestruko bolji performans kod rešavanja kompleksnih problema, i implementira Rete mrežu kod ulančavanja unazad. Zvanično je implementiran u CLIPS-u. Zatim imamo Rete III – koji je razvijen početkom 2000-ih, razvijen u saradnji sa inženjerima FICO kompanije, a u suštini predstavlja Rete II algoritam implementiran kao deo FICO Advisor engine-a. Konačno, Rete-NT – razvijen 2010, oko 500 puta brži od originalnog algoritma, i 10 puta brži od svog prethodnika, Rete II.



Slika 11 - Grafički prikaz Rete algoritma

2.9 Poslovne web aplikacije

Web aplikacije su postale od ključnog značaja za većinu kompanija u trenutnom visoko kompetitivnom okruženju. Web aplikacija predstavlja klijentsko-serverski softver koji koristi web pretraživač za prikazivanje interfejsa, i server za čuvanje podataka. Svi mogu da pristupe web aplikaciji sa bilo kog računara koji je povezan na internet, putem standardnih pretraživača. Većina ljudi je naviknuta na, na primer Microsoft Word, koji radi samo ako je instaliran na računaru. Međutim, sve što može biti urađeno u Microsoft Word-u može takođe biti urađeno preko neke web aplikacije, na primer Google Docs-a. Ukratko, poslovne web aplikacije omogućuju upravljanje i snimanje svih internih i eksternih operacija i procesa unutar kompanije.



Slika 12 - Arhitektura web aplikacije

Web aplikacije prate sledeći lanac komandi, prikazanih na slici 12: prvo, korisnik upućuje zahtev preko interfejsa na pretraživaču (tzv. *frontend*), koji se prosleđuje ka serverskoj mašini. Server zatim prosleđuje zahtev odgovarajućoj web aplikaciji (tzv. *backend*), nakon čega web aplikacija izvršava zadatak i generiše izveštaj. Web aplikacija potom šalje odgovor serveru, a server šalje procesuirane podatke pretraživaču, koji ih onda prikazuje korisniku.

Danas, skoro sve kompanije se odlučuju za korišćenje poslovnih web aplikacija. Razlozi su sledeći:

Bolja interoperabilnost – koristeći standardizovane web tehnologije, integrisanje web aplikacija postaje mnogo lakše. Instalacija i održavanje su jednostavni – potrebno je samo podići i održavati web aplikaciju na serveru da bi ona bila dostupna svim korisnicima. Dalje, poboljšana je bezbednost – većina web aplikacija je instalirana na namenskim serverskim mašinama, koje održavaju eksperti u oblasti administracije i bezbednosti. Ovo čini obezbeđivanje aplikacije jeftinim i pouzdanim. Bolji performans, jer moderni web serveri obično čuvaju web aplikacije na SSD-u i pokreću ih pomoću moćnih procesnih jedinica. U pogledu performansa, desktop aplikacije ne mogu da se mere sa web aplikacijama kada su u pitanju procesi koji zahtevaju mnogo hardverskih resursa. Zatim, raspoloživost je 24/7 – web aplikacije su obično hostovane na deljenim hosting servisima (npr. Google Cloud, Amazon Web Services – tzv. *cloud servisi*) koje pretplatnici plaćaju na osnovu resursa koje koriste. Skoro je nemoguće da ovi servisi ostanu bez hardverskih resursa za opsluživanje aplikacija. Isplativost, jer nakon što se jednom podigne na serveru, aplikaciji mogu da pristupe svi korisnici putem web pretraživača. Iako je potrebno testirati aplikaciju na više različitih pretraživača, razvoj je dovoljan da se radi na jednom operativnom sistemu, što čini aplikaciju isplativom u pogledu razvoja, isporuke i podrške. Potom fleksibilnost – pristup poslovnim platformama je lak bez obzira na to gde se korisnik nalazi, potrebno je samo da ima pristup internetu. Takođe, korisnička podrška je bolja, jer za razliku od desktop aplikacija gde se podrškom rukuje zasebno, kod web aplikacija nudi ugrađenu korisničku podršku kojoj korisnici mogu da pristupe onlajn. Konačno, dostupnost – dobro osmišljena i pažljivo dizajnirana web aplikacija pomaže kompanijama da lakše dođu do novih mušterija

Bitno je naglasiti da postoji razlika između web sajtova i web aplikacija. Web sajt uglavnom nudi statički sadržaj, i ograničenu funkcionalnost sa interaktivne tačke gledišta (eventualni izuzeci su forma za kontaktiranje ili polje za pretragu) i primarna uloga je prikaz informacija (npr. Wikipedia). Web aplikacije, sa druge strane, su dinamičke i interaktivne, i obično su zadužene za obavljanje kompleksnih zadataka – imaju poslovnu logiku u pozadini (npr Facebook, Skype, Donesi, Limundo itd).

2.10 Razvoj poslovnih web aplikacija

Već neko vreme, web pretraživači spajaju kompanije sa mušterijama, i vrlo je verovatno da će tako i ostati još dugo vremena. Iz tog razloga, neophodno je da zahtevi mušterija budu ispunjeni, poput bezbednosti, lakoće korišćenja, stabilnosti, itd. Uspešne i efikasne poslovne web aplikacije dele nekoliko karakteristika koje ih definišu: automatizuju i/ili eliminišu rutinske procese; agilne su, bezbedne, brze i razumljive krajnjim korisnicima.

Najlakši način da se postigne uspeh jeste da se prvo definišu krajnji ciljevi – na primer, kreirati listu procesa koje softver treba da izvršava, rutinskih operacija kojih se treba rešiti, i listu funkcija koje softver treba da ima. Potrebno je uključiti više aktera u proces diskusije, i dobaviti povratne informacije od što više krajnjih korisnika. Svi ovi zahtevi i preference mogu da se formalizuju sa takozvanom *listom zahteva*, koja će se poslati timu odgovornom za razvoj aplikacije.

Razvoj web aplikacije obično zahteva dve vrste ekspertize: *front-end development*, odnosno kodiranje sa klijentske strane – interpretiranje i izvršavanje koda na web pretraživaču, obično se razvija u tehnologijama kao što su Angular, Vue.js, React, itd. i *back-end development*, odnosno kodiranje sa serverske strane – interpretiranje i izvršavanje koda na samom serveru, obično se razvija u tehnologijama kao što su Java, Python, C#,... Što se tiče izbora baza podataka obično se koriste MySQL, Oracle, MongoDB, Redis, PostreSQL,...

Izbor radnog okvira i tehnologije je od krucijalnog značaja za razvoj poslovnih web aplikacija. Bez adekvatnog izbora, rizikuje se odlaganje isporuke aplikacije, i kompromitovanje kvaliteta koda.

Ceo proces razvoja može da se sumira u nekoliko koraka: prvi je definisanje strategije – dokumentovanje svrhe aplikacije, ciljevi i smer razvoja. Zatim sledi ispitivanje budućih korisnika aplikacije, kako bi se stvorila približna slika o tome kako aplikacija treba da izgleda. Sledi kreacija funkcionalnog dizajna aplikacije i izbor tehnologije, koja se uglavnom zavisi od specifikacije projekta definisane u funkcionalnom dizajnu. Potom se prelazi na dizajniranje vizuelnog izgleda aplikacije i definisanje strukture baze podataka. Tek nakon ovog koraka se prelazi na implementaciju – kodiranje. Nakon završetka implementacije sledi testiranje, dokaz o kvalitetu, pregledanje i isporuka, i na kraju, održavanje i podrška.

U fazi razvoja poslovnih aplikacija, bitno je spomenuti i cloud servise. Cloud servisi predstavljaju isporuku računarskih resursa i skladišnih kapaciteta kao uslugu za grupu krajnjih korisnika. Koncept cloud servisa se oslanja na deljenje resursa preko mreže, najčešće Interneta. Krajnji korisnici pristupaju aplikacijama u cloud-u preko web pretraživača ili desktop aplikacije na mobilnom telefonu, dok se softver i korisnički podaci nalaze na serverima na udaljenoj lokaciji. Ovaj model dozvoljava preduzećima da podignu i koriste aplikacije mnogo brže, sa boljom kontrolom i manje održavanja, što omogućava IT sektoru preduzeća da brže i efikasnije ispuni promenjive i nepredvidive zahteve poslovanja. Korisnik može samostalno da odabira i pokreće računarske resurse, može da bira vreme korišćenja i mrežni prostor za skladištenje podataka. Takođe, samouslužnim organizacijama omogućava stvaranje elastične okoline koja se povećava i smanjuje u zavisnosti od potrebe i ciljnih performansi. Računaski resursi provajdera usluga spajaju se kako bi poslužili sve korisnike koristeći model više zakupljenih jedinica (tzv. *Multi-Tenant model*), s različitim fizičkim i virtuelnim resursima, koji se dinamički dodeljuju i uklanjaju prema zahtevima korisnika. Primeri resursa uključuju mrežni prostor, procesore, memoriju, mrežnu propusnost, virtuelne mašine. Sistemi automatski proveravaju i optimizuju upotrebu resursa. To se postiže merenjem sposobnosti apstrakcije prikladne potrebnom tipu usluge (na primer skladištenje podataka, aktivni

korisnički računi). Upotreba resursa se može pratiti, proveravati i o njoj se mogu praviti izveštaji pružajući, tako, transparentan uvid o provajderima usluge i korisnicima. Mogućnosti koje korisnicima nudi cloud mogu biti ubrzano i elastično pokrenute, u nekim slučajevima i automatski, kako bi se po potrebi ostvarilo proporcionalno povećanje ili smanjenje resursa kada oni više nisu potrebni. Krajnjem korisniku resursi koje koristi mogu izgledati kao da nemaju ograničenja i mogu se kupiti u bilo kojoj količini u bilo koje vreme.

2.11 Ključni principi dizajna poslovnih web aplikacija

Alati za razvoj web aplikacija koje danas imamo na raspolaganju su izuzetno moćni i omogućavaju nam da razvijamo aplikacije koje su brze, visoko interaktivne i koje mogu da obavljaju izuzetno kompleksne zadatke. Međutim, prisustvo ovakvih alata ne znači automatski da ćemo napraviti aplikacije koje će ispuniti očekivanja i potrebe korisnika. Zapravo, veoma je lako zaneti se tehnologijom i zaboraviti principe interaktivnog dizajna, ostavljajući korisnika sa aplikacijom koja ne zadovoljava njegove potrebe.

Po Džeredu Spulu, postoji sedam osnovnih principa za dizajn efikasnih poslovnih web aplikacija:

1. Dizajn sa zadovoljavajućim detaljima – skoro svaka aplikacija zahteva od korisnika da napravi neki izbor. Odgovornost dizajnera je da obezbede dovoljno informacija kako bi korisnik napravio pravi izbor, a da pritom ne daju puno suvišnih informacija. Potrebno je opisati informacije jezikom bliskim korisniku, bez puno žargona i tehničkih izraza koje korisnik ne razume. Kada aplikacija od korisnika zahteva da napravi izbor, on mora jasno da zna koje su razlike među ponuđenim izborima. Ukoliko izbor uključuje neke dalje korake, oni moraju biti jasno naznačeni korisniku. Najbolji dizajneri se fokusiraju kako na sadržaj informacija, tako i na način na koji je informacija prezentovana (na primer, animacije u Angularu mogu da učine vizuelni doživljaj mnogo privlačnijim).

2. Informativan dizajn sa adekvatnim povratnim informacijama – bitno je obezbediti vizuelne naznake kako bi korisnici prepoznali neke napredne tehnike manipulacije sadržajem (na primer drag-and-drop, click-and-pull, itd.). Ovakve informacije mogu doći u različitim formama (pop-up dijalozi, animirane demonstracije, itd.) i timovi moraju da eksperimentišu kako bi videli koji način je najefektivniji za korisnike. Isto tako, dizajneri moraju da se pobrinu za to da korisnik ima informacije o tome šta se upravo desilo. Svaka bitnija operacija bi trebala da bude ispraćena odgovarajućom porukom koja bi korisnika obavestila o detaljima akcije koju je izvršio.

3. Dizajn u skladu sa pređašnjim iskustvima korisnika – dobri dizajneri se postaraju da imaju dovoljno informacija o pređašnjim iskustvima korisnika, i na osnovu njih dizajniraju svoje aplikacije. Na primer, u slučaju prikaza nekih tabelarnih podataka, ako dizajner zna da je korisnik iskusan sa Microsoft Excel-om, on će napraviti prikaz i interakciju sličnu Excelu u svojoj aplikaciji.

4. Fleksibilan dizajn – stvari ne idu uvek po planu, i korisnici ponekad naprave grešku u svojoj interakciji sa aplikacijom, ili ne dobiju rezultat koji su hteli. Iz tog razloga korisnicima treba omogućiti da eksperimentišu, i da ponište ono što su uradili (*undo*).

5. Defanzivan dizajn – obezbediti eksplicitne poruke greške na adekvatnim mestima ukoliko korisnik pokušava da izvrši nevalidnu akciju, i implementirati zaštitu protiv prekida rada. Na primer, često sajtovi tek nakon popunjavanja forme i potvrde iste javljaju da su neka polja nevalidna (lozinka ne ispunjava kriterijume, prekratko korisničko ime, itd.). U idealnom slučaju, potencijalna poruka greške bi se pojavljivala za vreme popunjavanja polja. Što se tiče zaštite protiv prekida rada, obično se implementiraju mehanizmi kao što je auto-save ili čuvanje sesije pretraživača.
6. Dizajn u skladu sa učestalošću korišćenja – neke aplikacije korisnik može koristiti jednom godišnje, a neke više puta dnevno. Korisnici koji često interaguju sa aplikacijom počinju da pamte terminologiju, lokacije funkcionalnosti, komande, i može biti frustrirajuće ukoliko je ovakve stvari teško upamtiti, ili ukoliko nema odgovarajućih prečica. Korisnici koji retko interaguju sa programom ne mogu da se oslone na memoriju i njima su potrebni vizuelni nagoveštaji. Zato dizajneri moraju da se konsultuju sa korisnicima i dobiju informacije o tome koliko će se često aplikacija koristiti, da bi u skladu sa tim mogli da rade dizajn.

7. Minimalistički dizajn – izbegavati dizajn elemente koji nisu neophodni. Zanimljivi efekti mogu da doprinesu dobrom dizajnu, ali ne treba preterivati.

2.12 Bezbednost poslovnih web aplikacija

Sajber kriminal je sve zastupljeniji i velike kompanije su često mete napada hakera. Nažalost, veliki procenat poslovnih web aplikacija ima sigurnosnih problema koji su lako rešivi. U većini slučajeva, sigurnosni problemi potiču od loših praksi, neulaganja u bezbednosna sredstva i zasterelih sistema.

U nastavku su navedene neke smernice za osiguravanje web aplikacija:

Neophodno je rigorozno testiranje ulaznih parametara – ulazni parametri su potencijalan put do osetljivih podataka i internih sistema. Ako softver povlači podatke sa Interneta, neko će eventualno pokušati da zaobiđe bezbednosne mere i prouzrokuje problem. Dobar primer za ovo je SQL injekcija. Na primer, na formi gde se traži unos poštanskog broja, karakteri se eventualno pretvaraju u SQL komandu. Ali, hakeri mogu da naprave takav unos da se on sam pretvori u nepredviđen SQL kod i izvrši se nad bazom. Iz tog razloga su bitne faze pregleda koda i testiranja, iako mogu da oduzmu mnogo vremena.

Korišćenje autentikacionih metoda bez lozinke – lozinke nisu toliko bezbedan mehanizam za čuvanje bitnih podataka, treba potražiti alternative, kao što je na primer N-faktor autentikacija.

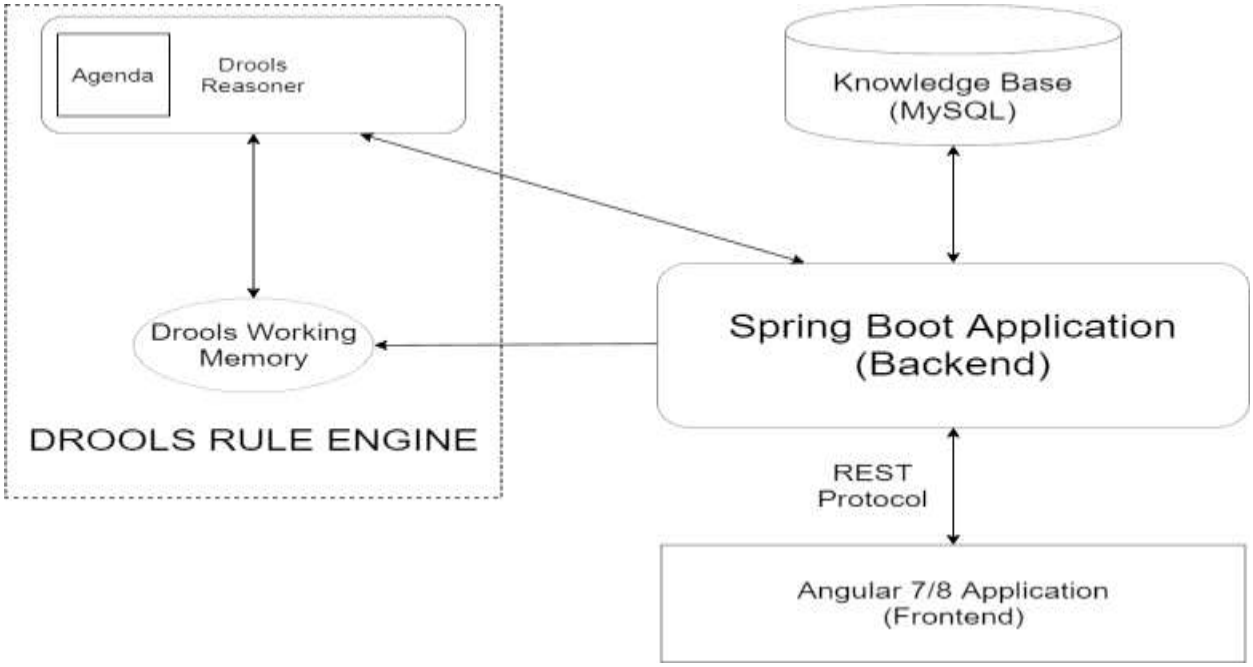
Takođe, poželjno je korišćenje WAF-a (Web Application Firewall) – štit između aplikacije i Internet saobraćaja, koji sprečava SQL injekcije, DDoS napade, uključivanje datoteka, cross-site skriptovanje i cross-site falsifikovanje. Cilj je da se zaustavi svaki potencijalno štetan web saobraćaj koji su drugi mehanizmi obezbeđenja propustili.

3 Arhitektura sistema

3.1 Struktura sistema

Studentski portal za mobilne studente je web aplikacija pisana u Javi, odnosno Javinom radnom okviru Spring Boot na serverskoj strani i Angularu 8 na klijentskoj strani i služi za kreiranje formulara od strane studenata koji žele da studiraju na fakultetu u inostranstvu, sa listom predmeta na domaćem fakultetu koje menjaju sa predmetima koje će slušati na stranom fakultetu. Komponente korisničkog interfejsa komuniciraju sa serverskom stranom preko REST protokola (*Representational State Transfer Protocol*). Kod ovog protokola, šalje se zahtev određenog tipa (koji tipično može biti GET, POST, PUT i DELETE, ali postoje i drugi), zahtev se šalje na resurs koji se jedinstveno identifikuje svojim URI-jem, i izvršava operaciju. Zahtev je obično praćen odgovorom, koji se vraća klijentskoj strani koja je pozvala REST operaciju. Telo REST zahteva mora imati tačno određeni format da bi se on uspešno izvršio. Ukoliko se operacija neuspešno izvrši, klijentu će biti poslat odgovor sa kodom i opisom greške.

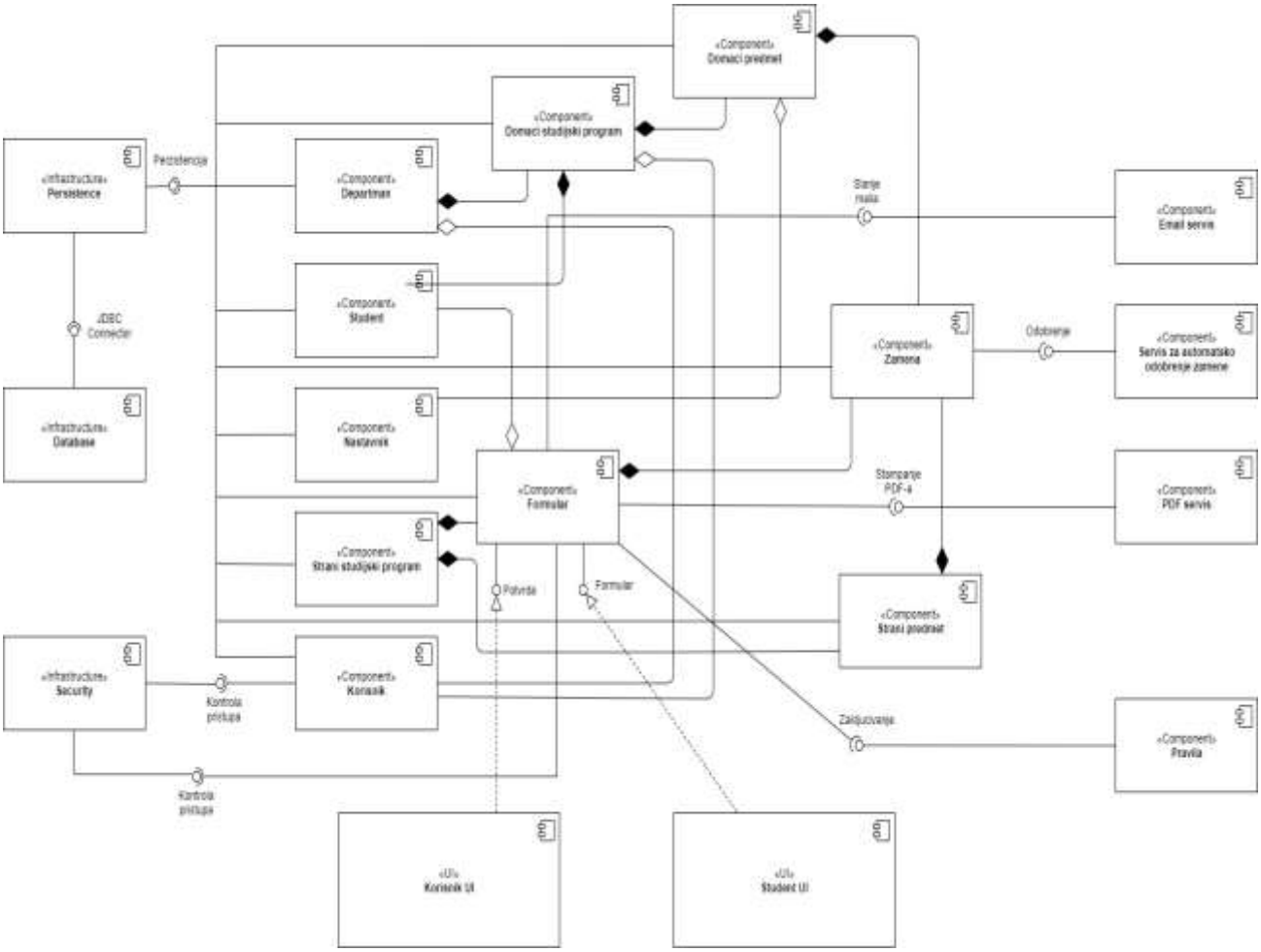
Dakle, svaka operacija unutar Spring Boot aplikacije je izložena kao REST servis koji se "gađa" sa adekvatnim zahtevom (na primer, ukoliko želimo da izlistamo sve formulare, poslaćemo GET zahtev sa praznim telom na URI http://localhost:8080/formulari, i aplikacija nam vraća odgovor u JSON formatu sa listom svih formuara). Nakon izvršenja operacije, dobićemo odgovor, i ukoliko operacija ima kao zadatak da kreira neki novi entitet, on će biti smešten u MySQL bazu podataka. Pravila se aktiviraju u momentu kada student kreira novi formular. On se u tom momentu ubacuje u radnu memoriju, a pravila će proći kroz sve zamene predmeta u formularu, proveriti da li su one validne na osnovu određenih kriterijuma i vratiti adekvatan odgovor klijentskoj strani. Nakon toga se formular briše iz radne memorije, i upisuje se u bazu podataka, ukoliko je validan. Opšta struktura sistema je prikazana na slici 13.



Slika 13 - Generalna struktura sistema

3.2 Arhitektura serverske strane

Na sledećem dijagramu prikazana je arhitektura komponenti sa serverske strane (slika 14):



Slika 14 - Arhitektura komponenti sa serverske strane

3.2.1 Departman

Na najvišem nivou hijerarhije se nalazi departman, sa sledećim poljima:

- departmanId – naziv departmana, takođe i primarni ključ
- koordinator – vezuje se za korisnika**,** određuje koordinatora departmana

Svaki departman ima jedan ili više domaćih studijskih programa. Koordinator departmana rukovodi isključivo onim aktivnostima koje se dešavaju u sklopu njegovog departmana.

3.2.2 Domaći studijski program

Odnosi se na studijske programe u sklopu domaćeg fakulteta. Ima sledeća polja:

- naziv – naziv studijskog programa, takođe i primarni ključ
- departman – vezuje se za departman, određuje kom departmanu pripada
- šef – vezuje se za korisnika, određuje šefa departmana

Svaki domaći studijski program se nalazi u sklopu jednog departmana, ima jednog ili više studenata koji na njemu studiraju, i sadrži jedan ili više domaćih predmeta. Šef studijskog programa rukovodi isključivo onim aktivnostima koje se dešavaju u sklopu njegovog studijskog programa.

3.2.3 Strani studijski program

Odnosi se na studijske programe u sklopu stranog fakulteta. Ima sledeća polja:

- naziv – naziv studijskog programa, takođe i primarni ključ

Svaki strani studijski program sadrži jedan ili više stranih predmeta. Razlika u odnosu na domaći studijski program je u tome što nisu relevantni departman i šef koji njime rukovodi, pa su ta ova polja ovde izbačena. Pri kreiranju formulara, bira se jedan od stranih studijskih programa na kom će student slušati nastavu.

3.2.4 Domaći predmet

Odnosi se na predmete u sklopu nastave domaćeg studijskog programa na domaćem fakultetu. Ima sledeća polja:

- predmetId – identifikaciona oznaka predmeta, takođe i primarni ključ
- naziv – naziv predmeta
- program – domaći studijski program u čijem sklopu se predmet nalazi
- espb – broj ESPB poena koje predmet nosi
- nastavnik – nastavnik koji predaje taj predmet

Domaći predmet pripada samo jednom studijskom programu, i drži ga samo jedan nastavnik. Pri kreiranju formulara, domaći predmet se menja sa stranim predmetom koji će se slušati na odabranom studijskom programu na stranom fakultetu.

3.2.5 Strani predmet

Odnosi se na predmete u sklopu nastave stranog studijskog programa na stranom fakultetu. Ima sledeća polja:

- predmetId – identifikaciona oznaka predmeta, takođe i primarni ključ
- naziv – naziv predmeta
- program – strani studijski program u čijem sklopu se predmet nalazi
- espb – broj ESPB poena koje predmet nosi

Strani predmet pripada samo jednom studijskom programu, a razlika u odnosu na domaći je u tome što je polje nastavnik izbačeno jer u ovom slučaju nije relevantno. Pri kreiranju formulara, domaći predmet se menja sa stranim predmetom koji će se slušati na odabranom studijskom programu na stranom fakultetu.

3.2.6 Student

Odnosi se na studente koji studiraju na domaćem fakultetu i koji će potencijalno napraviti formular za studiranje na stranom fakultetu. Ima sledeća polja:

- brIndeksa – broj indeksa studenta, u isto vreme i primarni ključ
- ime – ime studenta
- prezime – prezime studenta
- jmbg – jmbg studenta
- datumrodjenja – datum rođenja studenta
- email – email adresa studenta
- studije – domaći studijski program na kom student studira

Student studira na jednom domaćem studijskom programu. Student može da kreira formular gde bira strani studijski program na kom će slušati nastavu, i menja domaće predmete sa stranim. Student može imati samo jedan aktivan formular, a može i da obriše postojeći formular i da napravi novi.

3.2.7 Nastavnik

Odnosi se na nastavnike koji su zaposleni na domaćem fakultetu. Ima sledeća polja:

- nastavnikId – identifikaciona oznaka nastavnika, u isto vreme i primarni ključ
- ime – ime nastavnika
- prezime – prezime nastavnika
- jmbg – jmbg nastavnika
- datumrodjenja – datum rođenja nastavnika
- email – email adresa nastavnika

Nastavnik drži nijedan ili više domaćih predmeta na domaćem fakultetu. Takođe, nastavniku stiže email koji ga obaveštava da je predmet koji on drži zamenjen u formularu za neki strani predmet na stranom fakultetu. U mejlu se nalazi link preko kog nastavnik ide na formu na kojoj može da prihvati ili odbije zamenu svog predmeta. Nastavnik se ne loguje direktno na sistem i jedinu formu interakcije sa aplikacijom ima preko linka koji dobije u mejlu i koji ga vodi na formu gde on može da potvrdi ili odbije zamenu. Link sadrži jedinstveni identifikacioni token bez kog potvrda zamene nije validna, kako bi se sprečilo neautorizovano odobravanje zamene.

3.2.8 Korisnik

Odnosi se na korisnike sistema koji se direktno loguju na aplikaciju. Sadrži sledeća polja:

- username – korisničko ime, takođe i primarni ključ
- password – šifra korisnika
- ime – ime korisnika
- prezime – prezime korisnika
- jmbg – jmbg korisnika
- datumrodjenja – datum rođenja korisnika
- uloga – uloga korisnika, može biti admin/šef/koordinator

Korisnik može da izvrši logovanje na početnoj stranici unoseći korisničko ime i šifru i u zavisnosti od uloge će biti prosleđen na odgovarajuću formu. Postoje tri uloge za korisnika: admin, šef studijskog programa, koordinator departmana. Admin ima pravo da dodaje, briše ili menja sve entitete u sistemu. Koordinator departmana može da rukovodi najviše jednim departmanom i on daje inicijalnu potvrdu validnosti formulara, nakon koje se šalju mejlovi svim predmetnim nastavnicima koji predaju domaće premete u sklopu formulara. Šef studijskog programa može da rukovodi najviše jednim studijskim programom i on daje finalnu potvrdu na one formulare koji su odobreni od strane svih predmetnih nastavnika (i samim tim koordinatora). Nakon ove potvrde se generiše PDF i šalje se email studentu sa PDF-om da je njegov formular prihvaćen.

3.2.9 Formular

Odnosi se na formulare za studiranje u inostranstvu koje popunjavaju studenti. Sadrži sledeća polja:

- idFormular – identifikaciona oznaka formulara, takođe i primarni ključ
- student – student koji je popunio formular
- programStrani – odabrani strani studijski program
- odobrenjeSef – odobrenje šefa studijskog programa
- odobrenjeKoord – odobrenje koordinatora departmana
- datum – datum i vreme kreiranja formulara
- zamene – lista zamena, odnosno parova domaći predmet-strani predmet

Formular se kreira od strane studenta tako što on odabere strani studijski program na kom želi da sluša nastavu. Student ne može imati više od jednog formulara. Nakon što je odabrao, takođe treba da izabere sve domaće predmete koje želi da zameni sa stranim predmetima koje želi da sluša, što predstavlja listu zamena (par domaći predmet-strani predmet). Pravila proveravaju da li je formular validan. Formular treba da bude odobren prvo od strane koordinatora departmana, nakon čega se šalju mejlovi svim predmetnim nastavnicima. Nastavnici preko linka u mejlu mogu da potvrde ili odbiju zamenu, a šef studijskog programa na kraju daje finalnu potvrdu na formular koji ima odobrenja od svih predmetnih nastavnika. Nakon ove potvrde se generiše PDF i šalje se email studentu sa PDF-om da je njegov formular prihvaćen.

3.2.10 Zamena

Odnosi se na zamene domaćih predmeta stranim predmetima, u sklopu formulara. Sadrži sledeća polja:

- idZamena – identifikaciona oznaka zamene, takođe i primarni ključ
- formular – formular na koji se zamena odnosi
- predmetDomaci – odabrani domaći predmet
- predmetStrani – odabrani strani predmet
- odobreno – odobrenje nastavnika

U formularu može biti jedna ili više zamena, i predstavlja par domaći predmet – strani predmet, odnosno naznačava koji domaći predmet se menja kojim stranim predmetom. Nastavnik koji predaje domaći predmet preko email linka može da prihvati ili odbije zamenu predmeta.

3.2.11 Obezbeđenje

Spring Security je komponenta ugrađena u Spring Boot radni okvir, koja služi za autentikaciju, autorizaciju i kontrolu pristupa. Logovanje se izvršava preko forme na klijentskoj strani, a Spring Security proverava da li su prosleđeni kredencijali validni. Ako jesu, generiše se autorizacioni token koji se vraća klijentskoj strani i on se prosleđuje uz svaki naredni zahtev, a pri odjavi se ovaj token briše. Token sadrži i informaciju o tome koja je uloga prijavljenog korisnika. Takođe, ova komponenta vrši enkripciju korisnikove šifre. Svaka REST operacija koju izvršava korisnik je obezbeđena na dva načina: proverava da li je korisnik ulogovan, i ako je ulogovan, proverava da li ima odgovarajuću ulogu za izvršenje operacije.

Na primer, akciju dodavanja novog predmeta ima samo ulogovani korisnik sa ulogom ADMIN, dok ostali korisnici tu akciju ne mogu da izvrše, ili akcija koordinatorskog odobrenja, koju može da izvrši samo ulogovani korisnik sa ulogom KOORDINATOR.

3.2.12 Email servis

Email servis služi za slanje mailova. On se, uz pomoć parametara unutar Springovog properties fajla kači na Google-ov email server, s tim da mora postojati nalog na Gmail-u preko kojeg će se mejlovi slati. Unutar Spring properties fajla se specificira email adresa (u našem slučaju to je ftnemailservice.diplomski@gmail.com), lozinka i host/port servera za slanje mejlova na koji se kačimo. Mejlovi se šalju u sledećim situacijama:

- Slanje maila studentu kada šef studijskog programa odobri/odbije formular (sa PDF-om u prilogu, u slučaju odobrenja)
- Slanje maila studentu kada koordinator departmana odobri/odbije novi formular
- Slanje maila nastavniku sa linkom za potvrdu zamene predmeta
- Slanje maila studentu kada nastavnik odobri/odbije zamenu u formularu

3.2.13 Servis za automatsko odobrenje zamene

Ukoliko nastavnik ne odobri ili ne odbije zamenu predmeta u roku od 3 dana, pretpostavlja se da je ta zamena odobrena. Svakih 30 minuta izvršava se kod (tzv. *Cron job*) koji proverava neodobrene zamene koje su starije od 3 dana, i automatski ih odobrava.

3.2.14 PDF servis

PDF servis služi za generisanje PDF fajla sa formom za potpisivanje i listom predmeta koje student menja i koje će slušati na stranom fakultetu. Nakon što svi nastavnici odobre zamene svojih predmeta, šef studijskog programa daje finalnu potvrdu na formular, i tada se generiše PDF koji će biti poslat studentu na email.

3.2.15 Perzistencija podataka

Za perzistenciju podataka se koristi Javin JPA interfejs – predstavlja specifikaciju za upravljanje relacionim podacima u Java aplikacijama. Omogućuje pristup i perzistenciju podacima koji su reprezentovani kao Java objekti – prati tzv. objektno-relaciono mapiranje (ORM). Možemo da specificiramo koje objekte ćemo da perzistiramo, kao i kako ćemo da ih perzistiramo. Dakle, svaki entitet u aplikaciji je reprezentovan preko Javinih objekata. JPA nam nudi posebne anotacije sa kojima obeležavamo klase i polja tih klasa, i na taj način se kreira mapiranje između baze podataka i objekata unutar aplikacije. Takođe, svaki entitet ima svoj repozitorijum, koji predstavlja JPA interfejs sa gotovim metodama za pristup i perzistenciju entiteta (npr iščitaj sve, iščitaj jednog, sačuvaj jednog, itd.) i postoji mogućnost pisanja dodatnih metoda ukoliko je to potrebno.

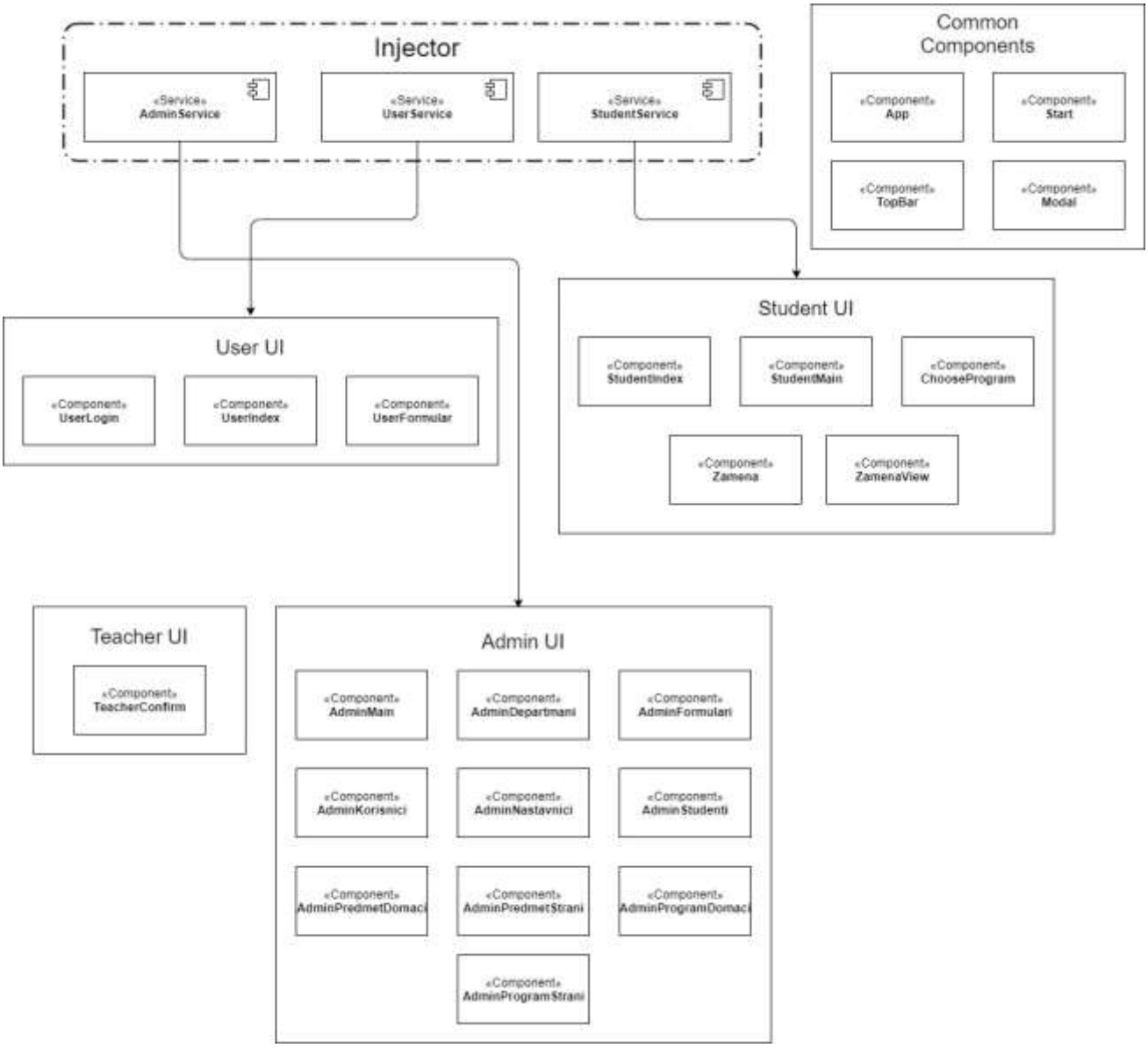
3.2.16 Rukovanje pravilima u aplikaciji

Pravila su pisana u jeziku za pravila Drools, i služe za proveru validnosti studentskog formulara. Nakon što student prosledi formular (koji u ovom slučaju predstavlja činjenicu), on se ubacuje u radnu memoriju Drools-a i nad njim se izvršavaju dva pravila: prvo se proveri da li je ukupan broj ESPB bodova domaćih predmeta veći od određenog broja, i proverava se svaka zasebna zamena, pri čemu broj ESPB bodova domaćeg predmeta mora biti manji ili jednak broju ESPB bodova predmeta koji će student slušati na stranom fakultetu. Ukoliko ove dve provere prođu, formular se smešta u agendu i označava kao validan, nakon čega se upisuje se u bazu. U suprotnom, student dobija odgovarajuću poruku greške. Nakon provere, formular se briše iz radne memorije Drools-a.

U svakom momentu u radnoj memoriji može da postoji najviše jedan formular, koji se briše na kraju procesuiranja, kako bi se izbeglo preopterećenje radne memorije. Različiti formulari nisu međusobno zavisni, pa nema potrebe da formular nakon procesuiranja ostane u radnoj memoriji.

3.3 Arhitektura klijentske strane

Na sledećem dijagramu prikazana je arhitektura komponenti sa klijentske strane (slika 15):



Slika 15 - Arhitektura komponenti sa klijentske strane

3.3.1 App komponenta

Korenska komponenta aplikacije, koja služi kao roditeljska komponenta za sve ostale komponente. Znači, sve ostale komponente koje budu prikazane će neizbežno biti u sklopu App komponente.

3.3.2 Servisi

Servisi u Angular aplikacijama obično služe za obavljanje diskretnih operativnih procesa kao što su pozivi ka serverskoj strani, manipulacija podacima, validacija i slično. Servisi su raspoloživi na nivou cele aplikacije. Svaka komponenta može da koristi funkcionalnosti definisane u servisu, s tim da je potrebna "injekcija" servisa u komponentu. U ovoj aplikaciji postoje tri servisa: Admin servis, User servis i Student servis. Admin servis čuva sve podatke relevantne za admina (podaci o studentima, korisnicima, nastavnicima, studijskim programima, predmetima,...), kao i podatke o ulogovanom adminu. User servis čuva podatke o ulogovanom korisniku, kao i sve formulare relevantne za tog korisnika. Student servis čuva podatke o studentu koji popunjava formular, kao i sve podatke o formularu koji popunjava, ili kojeg je već popunio.

3.3.3 Start komponenta

Komponenta koja se prikazuje pri pokretanju aplikacije, gde možemo da izaberemo da li želimo da se ulogujemo kao korisnik, ili da popunjavamo formular kao student.

3.3.4 TopBar komponenta

Header pri vrhu koji je uvek prikazan, i sadrži dugme za odlazak na početnu stranicu, kao i dugme za odjavu ukoliko je korisnik prijavljen.

3.3.5 Modal komponenta

Komponenta za rukovanje modalnim dijalozima na nivou cele aplikacije. Modalni dijalozi se prikazuju su slučaju kada, na primer, želimo da unesemo novog studenta (ili izmenimo postojećeg) ako smo ulogovani kao admin, i ova komponenta je zadužena za prikaz dijaloga i prikazivanje odgovarajućih polja u njemu.

3.3.6 TeacherConfirm komponenta

Komponenta za odobravanje/odbijanje zamene u sklopu formulara, kojoj se pristupa preko linka u email-u kog dobija nastavnik. Za pristup ovoj komponenti nije potrebna prijava na sistem, samo validan link.

3.3.7 UserLogin komponenta

Komponenta sa formom za prijavu korisnika, gde korisnik unosi svoje korisničko ime i šifru. Provera se vrši na serverskoj strani, i ukoliko je uspešna, korisniku se vraća autorizacioni token koji sadrži enkriptovane njegove kredencijale i ulogu. Nakon uspešnog logovanja, korisnik se preusmerava na odgovarajuću stranicu (u zavisnosti od uloge)

3.3.8 UserIndex komponenta

Početna stranica za korisnika, na koju se preusmerava nakon uspešne prijave, i na kojoj vidi informacije o svom nalogu.

3.3.9 UserFormular komponenta

Stranica na kojoj korisnik (šef departmana ili koordinator) vidi podatke o svim relevantnim formularima, koje može da odobri ili odbije, nakon čega se inicira slanje maila studentu i/ili nastavnicima.

3.3.10 StudentIndex komponenta

Stranica na kojoj student unosi svoj broj indeksa koji se validira sa serverske strane, kako bi mogao da popuni formular za studiranje u inostranstvu.

3.3.11 StudentMain komponenta

Početna stranica za studenta, na koju se preusmerava nakon što unese svoj broj indeksa i na kojoj vidi informacije o svom nalogu.

3.3.12 ChooseProgram komponenta

Stranica na kojoj student bira strani studijski program na kom želi da studira, iz liste ponuđenih programa.

3.3.13 Zamena komponenta

Stranica na kojoj student popunjava formular: konkretne zamene predmeta. Student treba da izabere najmanje jedan predmet sa studijskog programa na kom trenutno studira, na domaćem fakultetu i jedan predmet sa stranog studijskog programa na kom je izabrao da studira. Na serverskoj strani se vrši provera validnosti formulara.

3.3.14 ZamenaView komponenta

Detaljan pregled aktivnog formulara. Student može imati najviše jedan aktivan formular, i ukoliko pokuša da napravi novi a da pritom stari i dalje postoji, biće preusmeren na ovu stranicu. Opciono, na ovoj stranici student može i da obriše svoj aktivan formular.

3.3.15 AdminMain komponenta

Početna stranica za administratora sistema, na koju se preusmerava nakon uspešne prijave, i na kojoj vidi informacije o svom nalogu.

3.3.16 AdminDepartmani komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje departmana fakulteta.

3.3.17 AdminFormulari komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje studentskih formulara, kao i njihovih odgovarajućih zamena.

3.3.18 AdminKorisnici komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje korisnika sistema.

3.3.19 AdminNastavnici komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje nastavnika koji rade na domaćem fakultetu.

3.3.20 AdminStudenti komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje studenata koji studiraju na domaćem fakultetu.

3.3.21 AdminPredmetiDomaci komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje predmeta na domaćem fakultetu.

3.3.22 AdminPredmetiStrani komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje predmeta na stranom fakultetu.

3.3.23 AdminProgramiDomaci komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje studijskih programa na domaćem fakultetu.

3.3.24 AdminProgramiStrani komponenta

Komponenta na kojoj administrator sistema može da vrši dodavanje, izmenu ili brisanje studijskih programa na stranom fakultetu.

4 Implementacija

4.1 Odabrane tehnologije

4.1.1 Serverska strana – Spring Boot

Sa serverske strane, aplikacija je implementirana u Spring Boot radnom okviru, koji je razvijen od strane kompanije Pivotal. Prvenstveno pruža Java programerima platformu za razvijanje automatski konfigurabilnih aplikacija. Programeri mogu brzo i efikasno da razvijaju svoje aplikacije, bez gubljenja vremena na pripremu aplikacije i konfigurisanje. Spring Boot dolazi sa mnogo zavisnosti koje mogu da se uključe u aplikaciju (kao na primer biblioteka Spring Kafka, koja omogućava integrisanje aplikacije sa Kafka platformom, ili Spring Security, koji omogućava obezbeđivanje aplikacije od neautorizovanog korišćenja) i fokusira se na skraćivanju dužine koda i pružanju lakog načina za pokretanje aplikacije. Spring Boot je danas jedan od najpopularnijih radnih okvira za razvijanje serverskog koda aplikacije. Glavni razlog je što koristi Javu, jedan od danas najrasprostranjenijih programskih jezika. Pored toga, lako se uči, i društvo korisnika je veliko i pruža dobru podršku za probleme koji se javljaju u razvoju aplikacija. Neki od dodatnih benefita korišćenja Spring Boot-a su: smanjenje vremena razvoja aplikacija i povećanje produktivnosti razvojnog tima, omogućuje automatsku konfiguraciju svih komponenti, smanjenje repetitivnog koda, zatim dolazi sa ugrađenim HTTP serverima kao što su Tomcat ili Jetty i nudi veliki broj dodataka koji omogućuju povezivanje sa drugim sistemima, kao što je MySQL baza podataka, ElasticSearch, ActiveMQ,...

4.1.2 Baza podataka – MySQL

MySQL je besplatna relaciona baza podatka sa klijent-server modelom, razvijena od strane švedske kompanije MySQL AB. Podaci su, kao i kod svih relacionih baza, strukturirani i organizovani po tabelama koje mogu biti međusobno povezane. Za komunikaciju sa bazom se koristi domenski specifičan jezik pod nazivom SQL (Structured Query Language). Jedan ili više uređaja (klijenata) uspostavlja konekciju sa serverom i svaki od njih može da uputi zahtev za pristup podacima ka serveru, i ukoliko je zahtev validan, server šalje željene podatke.

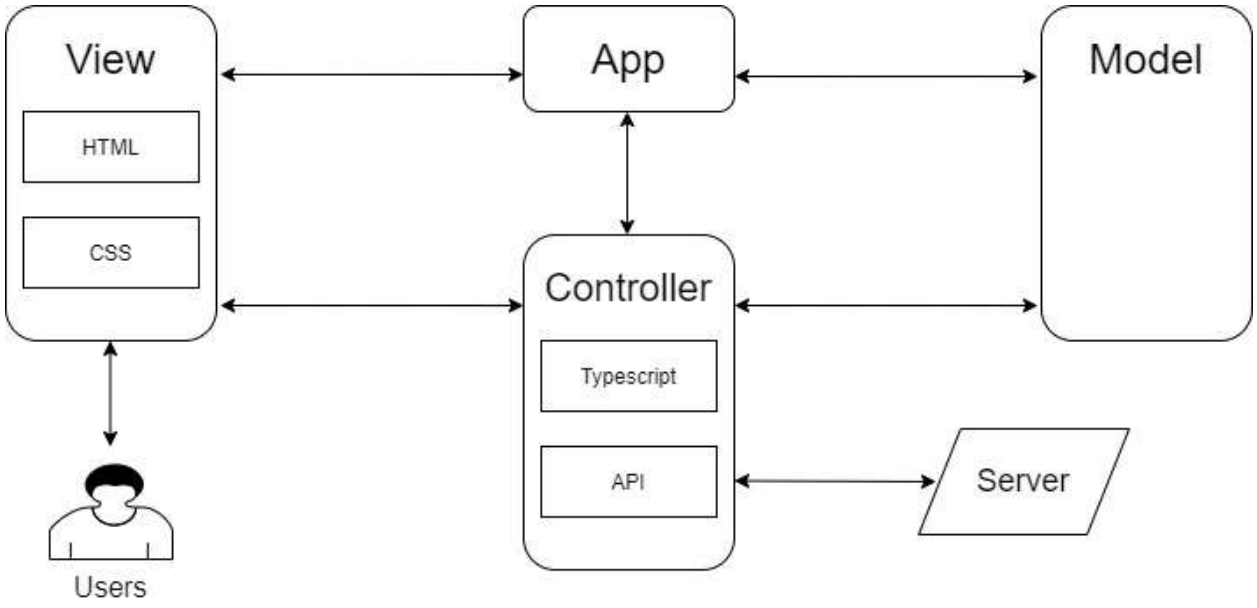
MySQL je jedna od najpopularnijih opcija kada su u pitanju relacione baze, nalazi se na drugom mestu odmah iza Oracle baza. Razlozi su sledeći: fleksibilnost i lakoća upotrebe – upotreba je laka i intuitivna, i postoji mogućnost menjanja koda. Zatim, predstavlja industrijski standard – MySQL koristi veliki broj kompanija širom sveta. MySQL odlikuje optimizovanost – server izuzetno brzo šalje odgovore na svaki zahtev, kao i bezbednost – nudi enkripciju lozinkom i verifikaciju mašine.

4.1.3 Klijentska strana – Angular 8

Angular je radni okvir napisan u JavaScript-u, jednoj od najpopularnijih tehnologija za pisanje web aplikacija, odnosno klijentskog koda web aplikacija – korisničkog interfejsa. JavaScript je u jednom momentu izgubio na popularnosti jer nije bio optimalan za razvoj kompleksnih korisničkih interfejsa, koji su sa vremenom bivali sve složeniji. Rešenje je bilo napraviti radni okvir koji će razdvojiti logiku aplikacije od manipulacije grafičkim komponentama – ovaj šablon se zove *model-view-controller*. U ovom šablonu imamo tri komponente, prikazane na slici 16: model, koji prestavlja podatke, zatim pogled (*view*), koji predstavlja grafički prikaz podatka i akcija koje se nad njima mogu izvršavati, i kontroler (*controller*), koji je zadužen za logiku aplikacije i interpretaciju korisničkih akcija.

Takođe, ono što je karakteristično za Angular je dinamičko osvežavanje stranica aplikacije, koje je omogućeno vezivanjem podataka (*data binding*). Pored toga, postoje i *direktive,* odnosno mogućnost kreiranja novih grafičkih komponenti od strane programera. Slično kao kod Spring Boot-a, u Angularu takođe postoje zavisnosti koje mogu da se uključe u aplikaciju. Tipično, Angular se koristi u sinergiji sa Node.js-om, koji nam pruža paket menadžer i web server koji čine razvoj Angular aplikacija lakšim.

Angular je u poslednjih par godina stekao veliku popularnost, a razlozi su sledeći: predstavlja poboljšani JavaScript – Angular koristi Typescript koji se oslanja na JavaScript, tako da programeri sa iskustvom u JavaScriptu lako savladaju Angular, zatim postoje dizajn šabloni za održivost – kod je dobro strukturiran i grupisan, pa ne treba utrošiti mnogo vremena da bi se razumeo. Takođe, uočava se razdvojenost komponenti – Angular uklanja čvrstu spregu između komponenti, ovo olakšava čitljivost, održivost i razumljivost. Testiranje je lako – uz pomoć dodatnih alata kao što su Jasmine i Protractor. Postoji mogućnost razvijanja kako desktop, tako i mobilnih aplikacija i ima aktivno održavanje – zajednica sa velikim brojem korisnika, uz konstantno održavanje

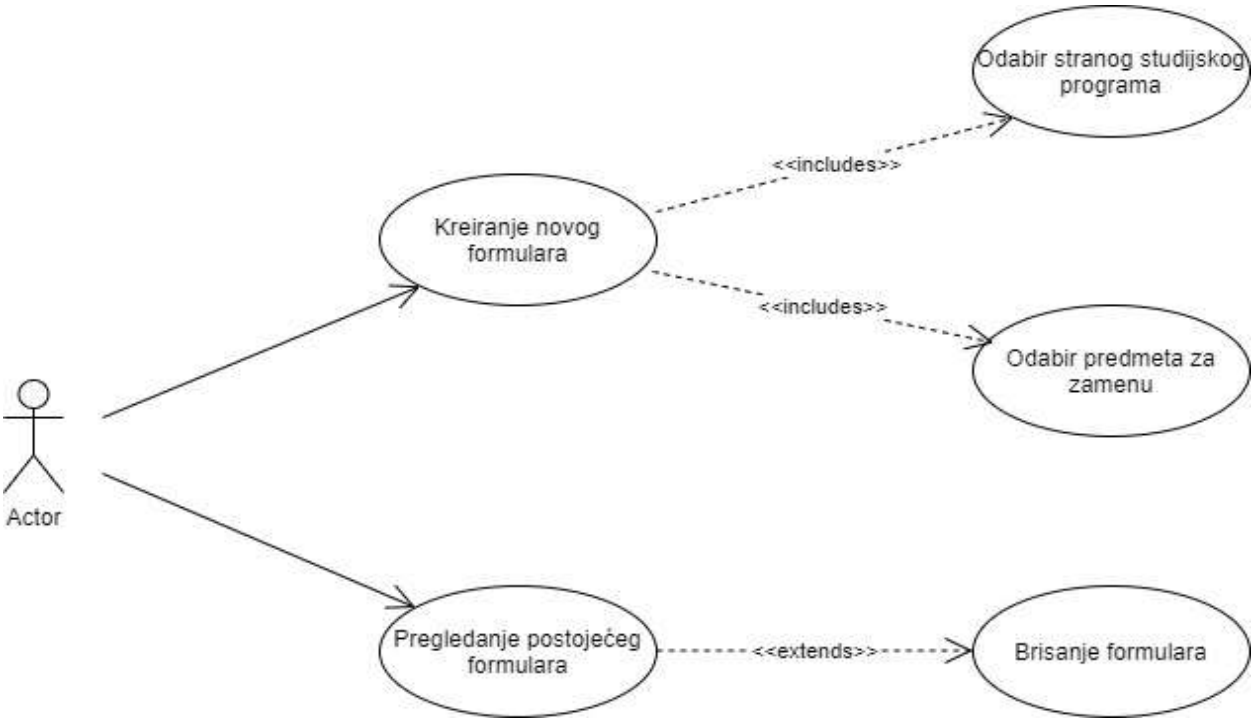


Slika 16 - Arhitektura Angular aplikacije

4.2 Slučajevi korišćenja i detalji implementacije

4.2.1 Student

Student u aplikaciji može da kreira formular za studiranje u inostranstvu, da pregleda postojeći formular, ili da poništi postojeći formular (slika 17). Student na početku vidi formu u koju unosi svoj broj indeksa, i ukoliko je on validan, biće preusmeren na stranicu gde može da vidi svoje podatke i dugme na koje može da klikne da bi popunio formular. Ukoliko student već ima formular, klik na ovo dugme će ga preusmeriti na stranicu gde može da vidi detalje o svom aktivnom formularu. Na ovoj stranici se takođe nalazi dugme za poništavanje formulara, i klikom na ovo dugme se formular briše iz baze podataka i student može da napravi novi formular.



Slika 17 - Dijagram slučajeva korišćenja za studenta

Nakon što student klikne na dugme za kreiranje formulara biće preusmeren na jednostavnu formu gde treba da izabere strani studijski program na kom želi da studira. Izbor stranih predmeta u zamenama će direktno zavisiti od odabira stranog studijskog programa. Nakon što je odabrao, biće preusmeren na formu gde može da bira zamene predmeta. Ova forma je prikazana tabelarno i postoji jedna kolona gde se bira predmet sa domaćeg fakulteta koji će biti zamenjen (ponuđeni predmeti su oni sa njegovog trenutnog studijskog programa), i jedna kolona gde se bira predmet sa stranog fakulteta koji će student slušati umesto odgovarajućeg domaćeg predmeta (ponuđeni predmeti su samo oni sa stranog studijskog programa kojeg je izabrao u prethodnom koraku). Ovaj par čini jednu zamenu. Student može napraviti najviše onoliko zamena koliko ima predmeta na domaćem ili stranom programu, u zavisnosti od toga koji ima manje predmeta. Nakon što je popunio sve zamene, student prosleđuje formular ka serverskoj strani, gde se aktiviraju pravila koja ga validiraju. Formular se ubacuje u radnu memoriju Drools-a, i postoje dva

pravila, gde jedno proverava ukupan broj ESPB bodova svih odabranih domaćih predmeta (mora biti ukupno više od 10 ESPB bodova), i drugo koje proverava validnost svake zamene – odnosno, da li je broj ESPB bodova stranog predmeta veći ili jednak ESPB bodovima domaćeg predmeta. Ukoliko je formular validan, student će biti obavešten o tome i biće sačuvan u bazi podataka uz pomoć JPA interfejsa, a ukoliko nije, student dobija informaciju o tome šta nije validno. Nakon provere, formular se uklanja iz radne memorije da bi se izbeglo njeno preopterećenje. Kad god se promeni status formulara ili zamene, student dobija obaveštenje na email – to podrazumeva odobrenje ili odbijanje od strane koordinatora departmana, predmetnog nastavnika i šefa studijskog programa (slika 18). Nakon što su svi nastavnici dali odobrenja o zameni svojih predmeta, šef može da da finalnu potvrdu za studiranje u inostranstvu, i studentu će na email stići obaveštenje sa potvrdom u vidu PDF-a u prilogu maila.

Slika 18 - Email koji obaveštava studenta o statusu zamene

U kodu imamo 5 REST operacija:

- POST /indexValidation – validacija unetog broja indeksa, ukoliko je validan i postoji u bazi, student se propušta na sledeću formu – početnu stranicu gde student vidi svoje podatke, a ukoliko nije ispisuje se poruka greške
- POST /api/podloga – provera da li student već ima aktivan formular, ukoliko ima preusmerava se na stranicu za pregled aktivnog formulara, a ukoliko nema, preusmerava se na formu za kreiranje novog formulara gde se bira strani studijski program (pri čemu se lista raspoloživih stranih studijskih programa vraća u telu REST odgovora)
- POST /api/podloga/straniProgram – potvrda odabranog stranog studijskog programa, u telu REST odgovora se vraćaju liste predmeta sa domaćeg studijskog programa i sa odabranog stranog studijskog programa
- POST /api/podloga/{id}/zamene – provera validnosti prosleđenog formulara sa zamenama. Ukoliko je validan, zamene i formular se perzistiraju u bazi i student dobija informaciju da je formular validan, nakon čega on čeka potvrde od strane koordinatora departmana, nastavnika i šefa studijskog programa. Ukoliko nije validan, student dobija informaciju o tome i formular neće biti sačuvan u bazi sve dok ne bude validan.
- DELETE /api/podloga/{id}/cancelForm – brisanje aktivnog formulara iz baze podataka, nakon čega student ponovo može da kreira svoj formular

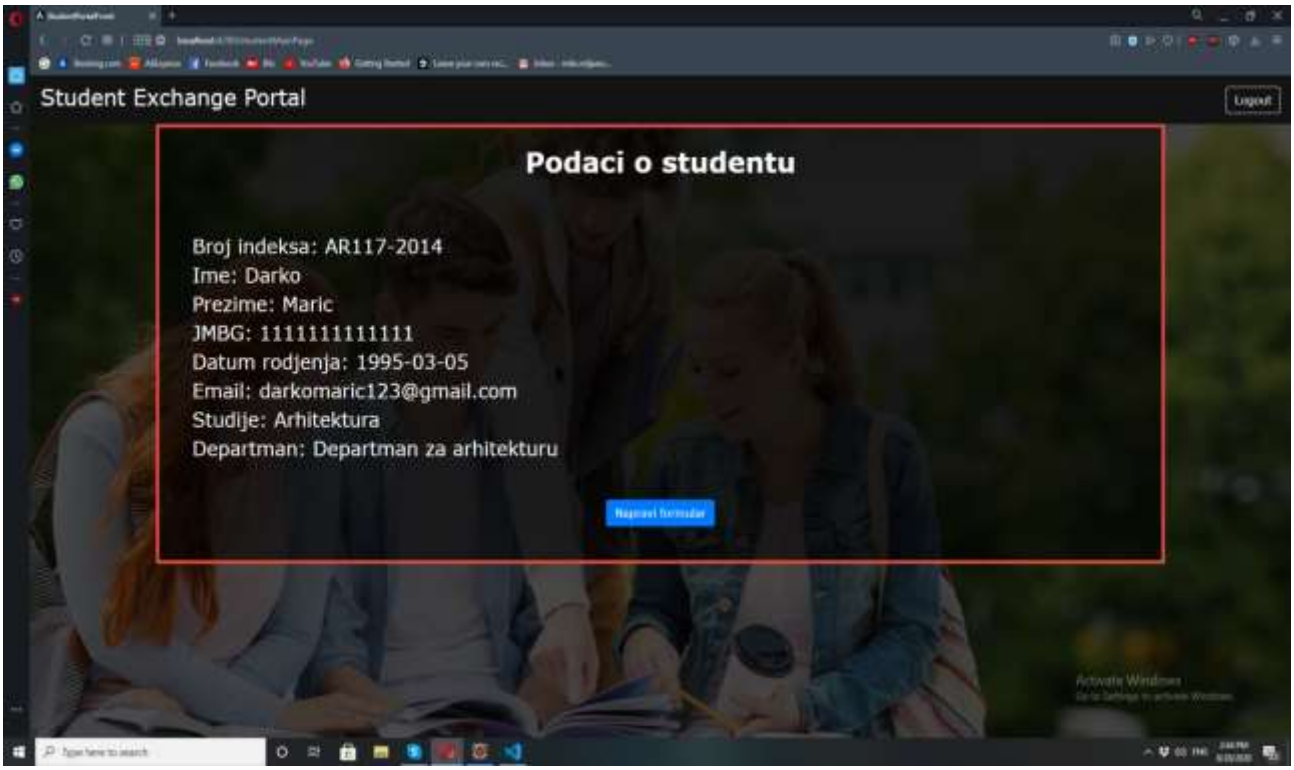
Što se tiče izvršavanja pravila, prvi korak je da u radnu memoriju ubacimo listu zamena, i novokreirani objekat koji reprezentuje formular, i koji će potencijalno biti sačuvan u bazi podataka, u slučaju da je validan. U prvom pravilu uzimamo oba ova objekta, i proveravamo da li svaka zamena u listi ispunjava uslov da je broj ESPB bodova domaćeg predmeta manji ili jednak ESPB bodovima stranog predmeta (provera se vrši tako što pravimo novu privremenu listu zamena koje ispunjavaju ovaj uslov i poredimo veličinu liste sa veličinom originalne liste – ukoliko su jednake, uslov je ispunjen). Potom proveravamo drugi uslov, a to je ukupan broj ESPB bodova svih domaćih predmeta. Ukoliko je on veći

od određene vrednosti (u ovom slučaju 10), formular je validan, i boolean polje valid u objektu se stavlja na true, što znači da će biti sačuvan u bazi i prosleđen odgovarajućim korisnicima aplikacije za potvrde. Nakon ovoga, izvršava se drugo pravilo, koje čisti listu zamena i formular iz radne memorije, jer oni više tamo nisu potrebni, i na ovaj način se izbegava potencijalno preopterećenje memorije. U nastavku se nalazi Drools kod opisanih pravila:

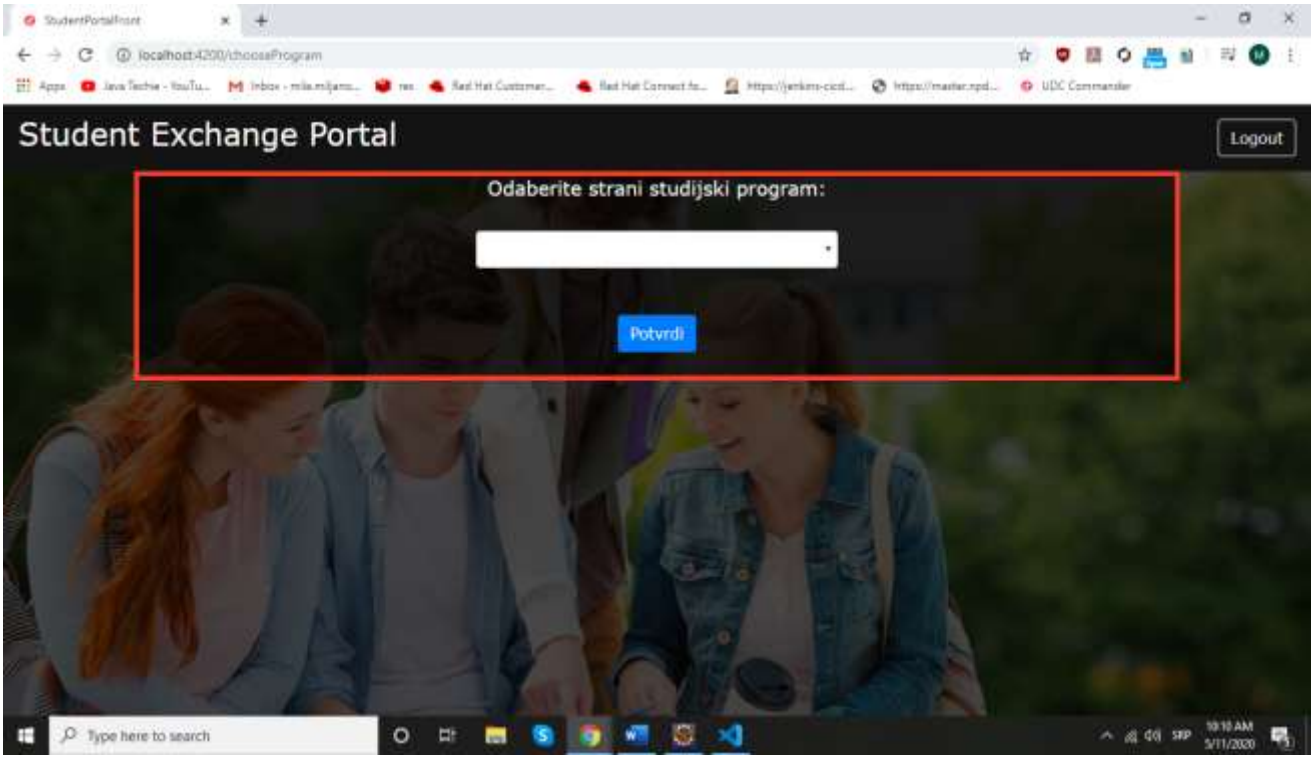
```
rule "ESPB validity"
agenda-group "valid"
      when
            $sfr: SubmitFormRequest( $zamene: zamene )
            $f: Formular ()
            $zl: List( size == $zamene.size() ) from accumulate (
                  $z: Zamena(predmetDomaci.espb <= predmetStrani.espb) from
$zamene,
                  collectList($z)
            )
            $totalEspb : Number( intValue > 10 ) from accumulate (
            Zamena($espb: predmetDomaci.espb) from $zamene,
            sum( $espb )
        )
      then
            modify ($f) { setValid(true) }
end

rule "Clear"
agenda-group "clear"
      when
            $sfr: SubmitFormRequest( )
            $f: Formular ()
      then
            retract ($sfr)
            retract ($f)
End
```

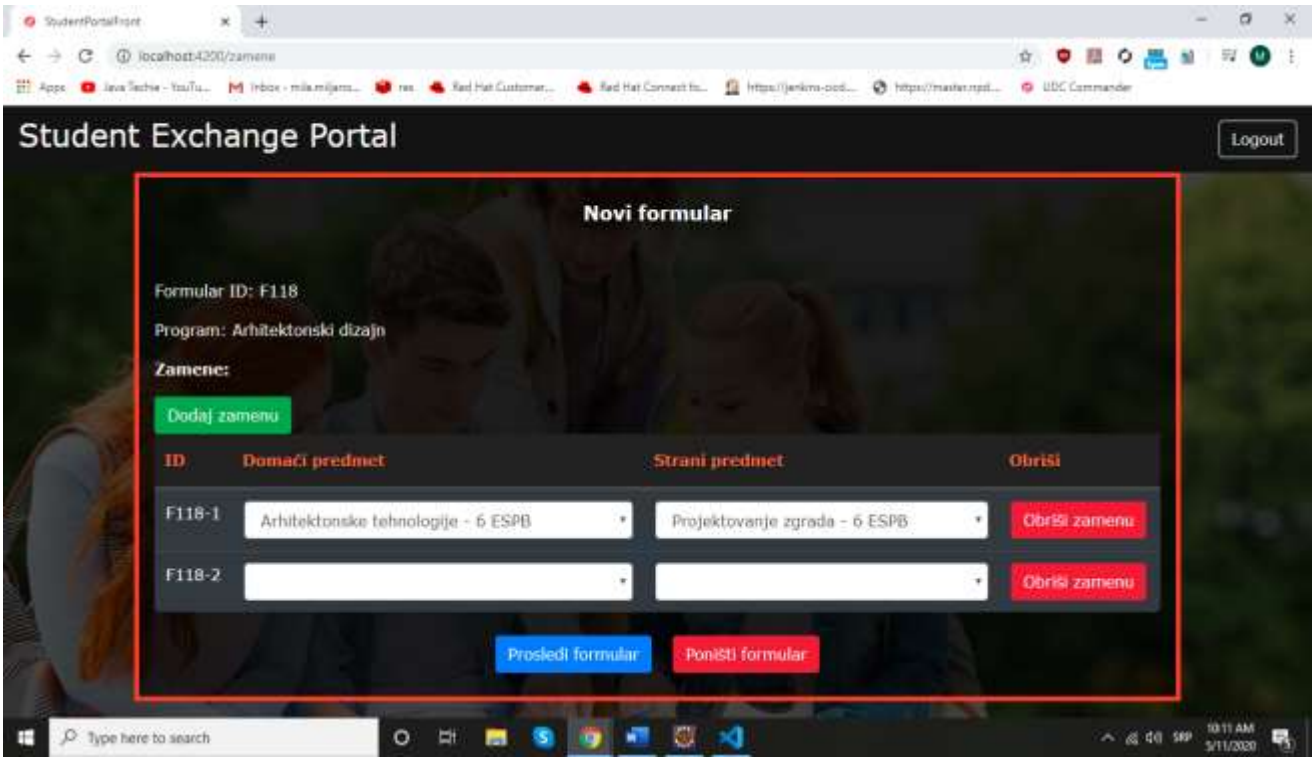
Izgled grafičkog interfejsa za studenta vidimo na slikama 19, 20, 21 i 22.



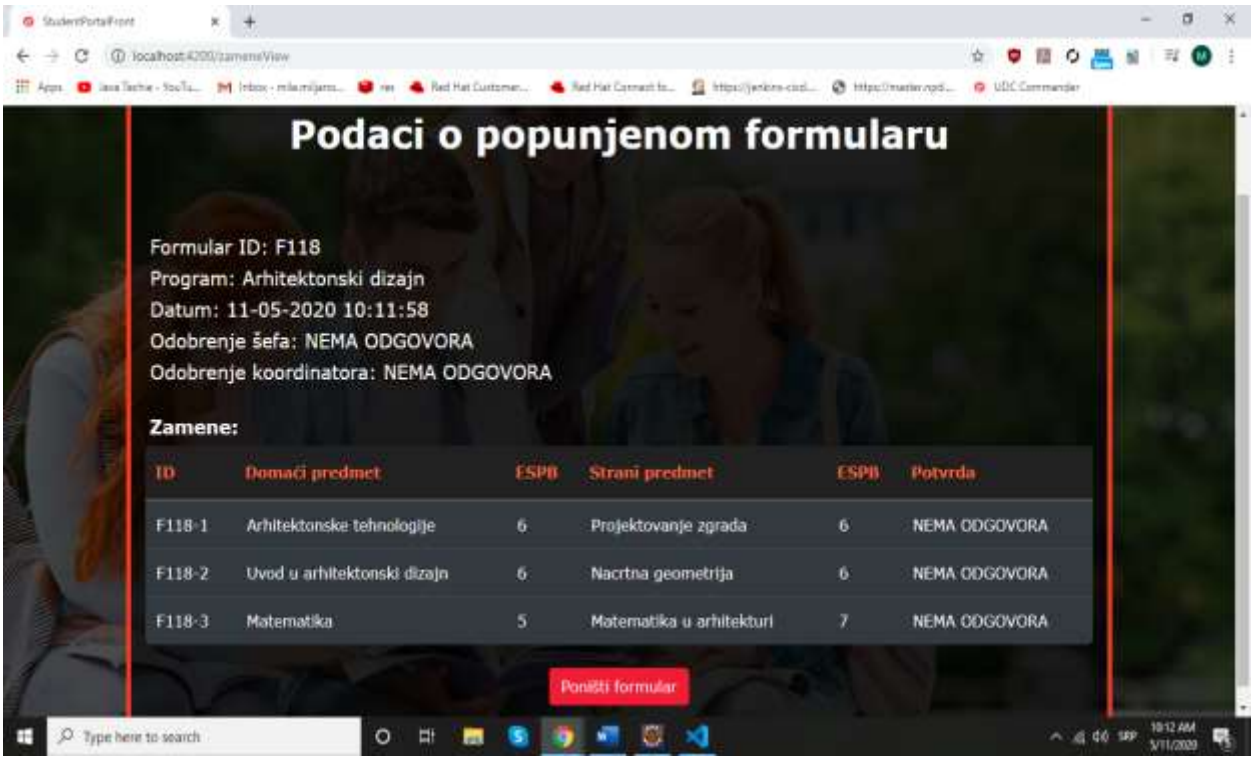
Slika 19 - Izgled grafičkog interfejsa za studenta – početna stranica



Slika 20 - Izgled grafičkog interfejsa za studenta – forma za biranje stranog studentskog programa



Slika 21 - Izgled grafičkog interfejsa za studenta – forma za popunjavanje zamena



Slika 22 - Izgled grafičkog interfejsa za studenta – pregled aktivnog formulara

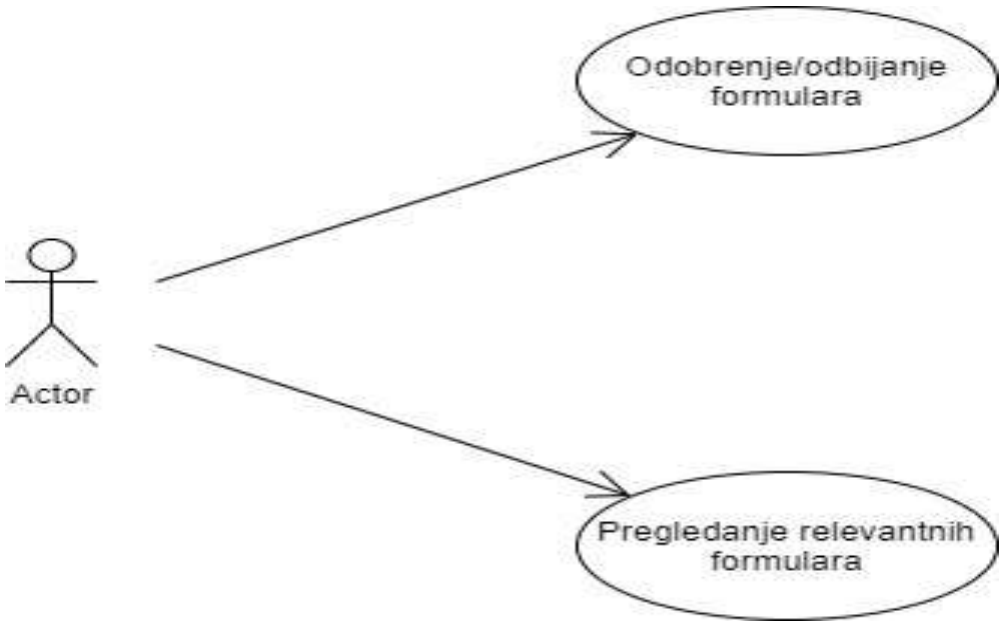
4.2.2 Korisnik i nastavnik

Postoje 3 vrste korisnika aplikacije: koordinator departmana, šef studijskog programa i nastavnik. Svaki od njih ispunjava određene zadatke, koji su navedeni u nastavku teksta.

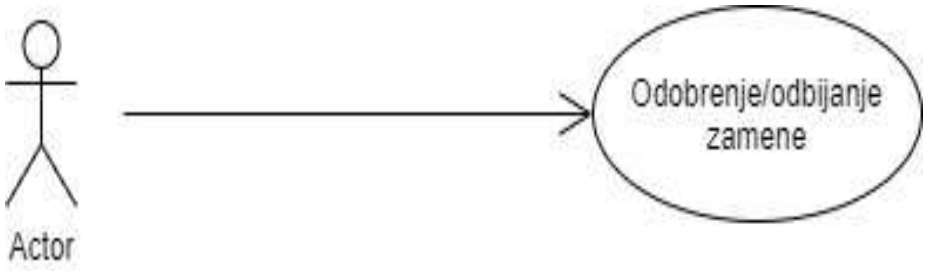
Koordinator departmana daje odobrenja na nove formulare, ali samo u sklopu departmana za koji je on odgovoran (slika 24). Da bi dao odobrenje, koordinator mora da se uloguje na sistem sa svojim kredencijalima, nakon čega on vidi listu svih novih formulara koji čekaju potvrdu sa njegove strane. Koordinator može da odobri ili odbije novi formular. Ukoliko odbije, student će dobiti informaciju na mejl o tome i on potom može da obriše taj formular i napravi novi. Ukoliko koordinator da odobrenje, student se o tome izveštava na email, i takođe se šalju mejlovi svim predmetnim nastavnicima čiji su predmeti izabrani za menjanje u tom formularu.

Šef studijskog programa daje odobrenja na formulare koji su odobreni od strane koordinatora i svih predmetnih nastavnika čiji se predmeti menjaju u sklopu tih formulara (slika 23). Šef vidi samo one formulare koji se tiču studijskog programa za koji je on odgovoran. Da bi dao odobrenje, šef mora da se uloguje na sistem sa svojim kredencijalima, nakon čega on vidi listu svih formulara koji čekaju potvrdu sa njegove strane. Šef može da odobri ili odbije formular. Ukoliko odbije, student će dobiti informaciju na mejl o tome i on potom može da obriše taj formular i napravi novi. Ukoliko šef da odobrenje, student se o tome izveštava na email, i u prilogu maila se šalje PDF potvrda o validnom formularu. Šef studijskog programa ne može da da potvrdu sve dok svi predmetni nastavnici ne daju svoje potvrde.

Nastavnik daje odobrenja na zamene u sklopu formulara, ali samo za one domaće predmete na kojima predaje (slika 24). Nastavnik se za razliku od ostalih korisnika ne loguje na sistem – po odobrenju koordinatora, njemu u mailu stiže link do forme na kojoj on može da da svoje odobrenje za odgovarajuću zamenu. Nakon što odobri ili odbije zamenu, student se o tome obaveštava na mail. Ukoliko nastavnik ne da odobrenje ili odbijanje u roku od 3 dana, zamena se smatra odobrenom i ona se automatski odobrava



Slika 23 - Dijagram slučajeva korišćenja za koordinatora i šefa



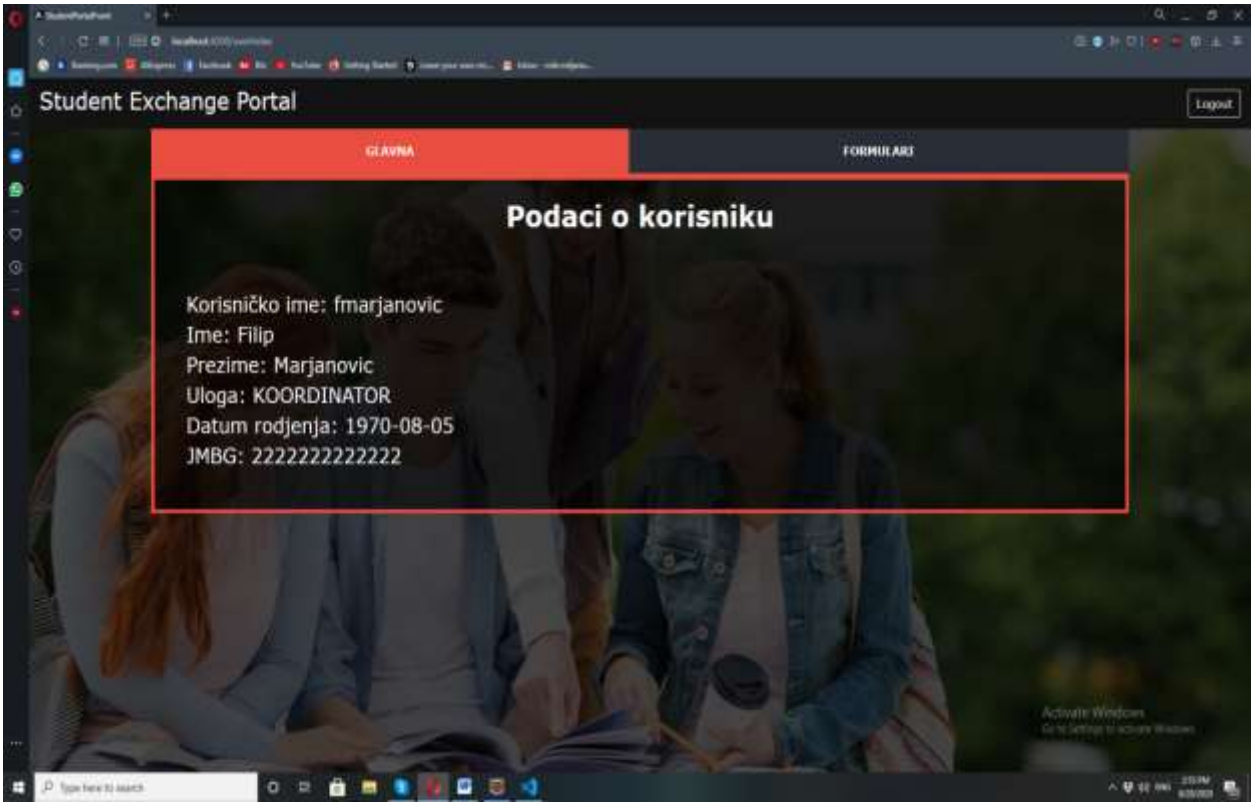
Slika 24 - Dijagram slučajeva korišćenja za nastavnika

Sve REST operacije koje se tiču koordinatora i šefa (i administratora) su obezbeđene uz pomoć Spring Security-ja, i to na dva načina: proverava se da li su kredencijali validni, i proverava se da li je uloga odgovarajuća. To znači da na primer, koordinator ne može da izvrši operaciju koja je predviđena za šefa ili administratora. Uloga i kredencijali se proveravaju iz autorizacionog tokena, koji se šalje sa klijentske strane uz svaki zahtev, a token se dobija nakon što se korisnik uspešno uloguje na sistem. Ova komponenta takođe vrši enkripciju korisnikove šifre u bazi podataka.

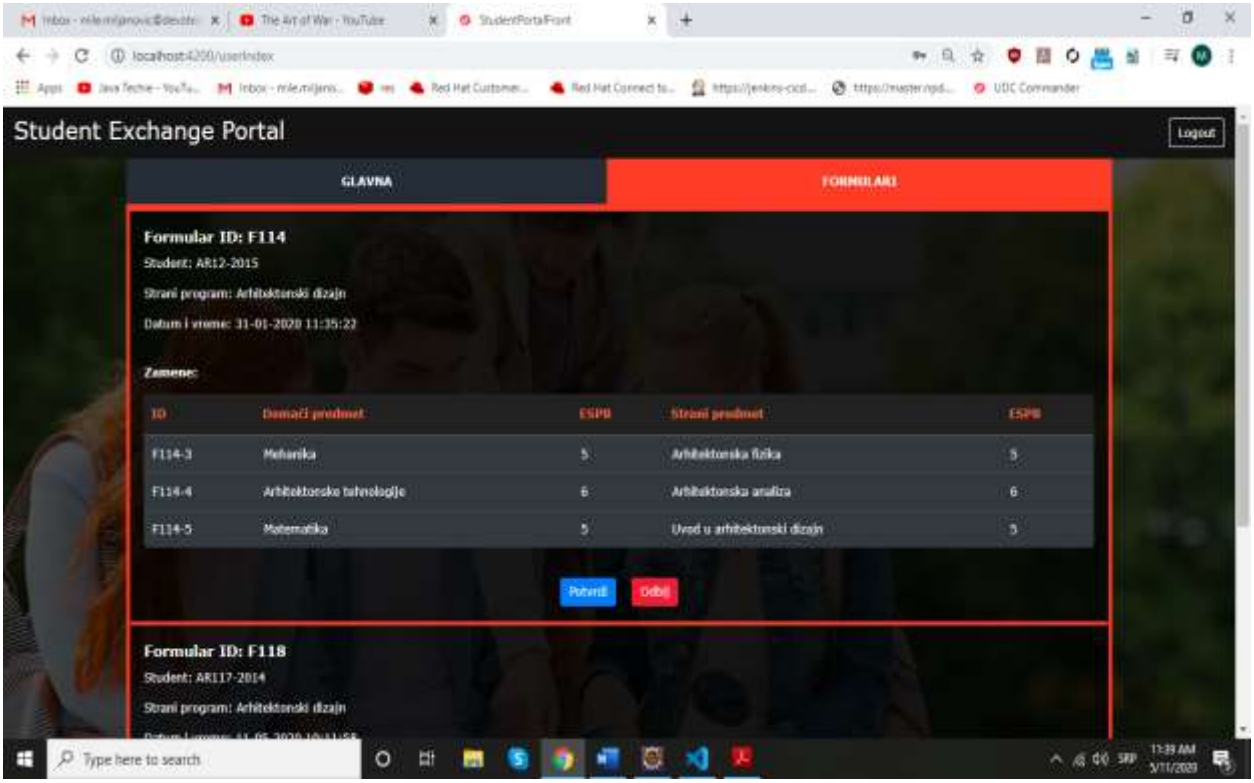
U kodu imamo 5 REST operacija:

- POST /login – logovanje korisnika, ukoliko su prosleđeni kredencijali validni, korisnik se propušta na narednu formu i vraća se autorizacioni token koji se koristi u svim narednim zahtevima, a ukoliko nisu korisnik dobija informaciju o neuspešnom logovanju
- POST /api/formulari – vraća listu relevantnih formulara: koordinatoru se vraća lista novih formulara bez odobrenja, a šefu se vraća lista formulara odobrenih od strane svih nastavnika
- PUT /api/formulari/{id}/koordinatorConfirm – odgovor koordinatora na novi formular: ukoliko je potvrdio šalju se mejlovi svim predmetnim nastavnicima sa linkovima, kao i obaveštenje studentu na mail. Ukoliko je odbio, samo student dobija obaveštenje o tome preko maila. Odgovor koordinatora se potom perzistira u bazi
- PUT /api/formulari/{id}/sefConfirm – odgovor šefa na formular sa svim potvrdama: ukoliko je potvrdio, šalje se mail studentu sa PDF-om u prilogu, sa formama za potpisivanje i listom svih zamena. Ukoliko je odbio, student dobija obaveštenje o tome preko maila. Odgovor šefa se potom perzistira u bazi
- GET /api/formulari/{formular}/zamene/{zamena}/{uuid}/{odgovor} – odgovor nastavnika na zamenu: nakon što da odgovor student se o tome obaveštava na mail. UUID u linku predstavlja jedinstveni token (koji se generiše pri kreiranju formulara i čuva u bazi) bez kog zamena ne može da se potvrdi. Validan token se prosleđuje u okviru linka u nastavnikovom emailu – na ovaj način je link obezbeđen od neautorizovanog pristupa. Ukoliko nastavnik ne da odgovor u roku od 3 dana, zamena se automatski odobrava

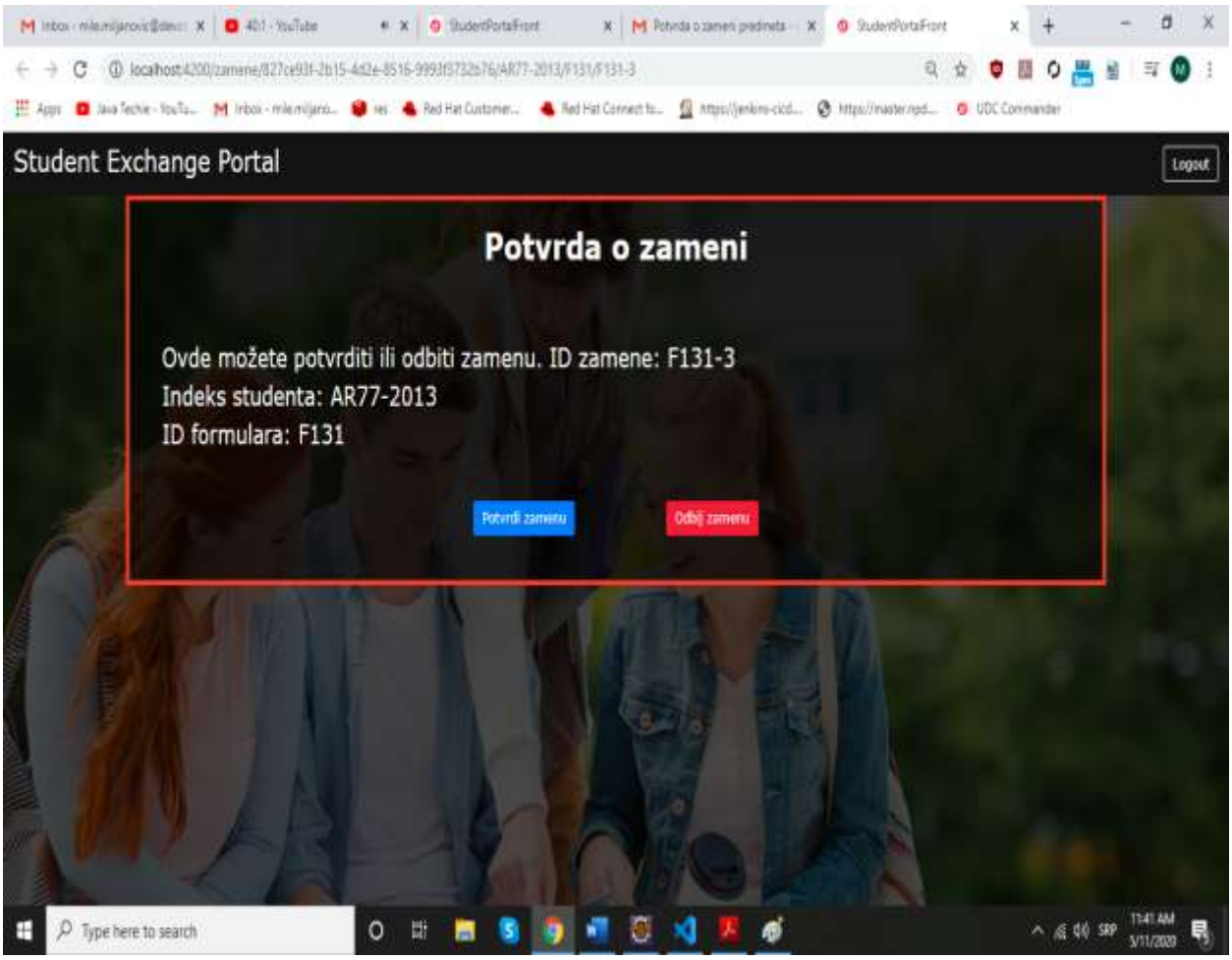
Izgled grafičkog interfejsa za korisnika vidimo na slikama 25 i 26, a za nastavnika na slici 27. Primer emaila za nastavnika se nalazi na slici 28, a izgled PDF dokumenta na slici 29.



Slika 25 - Izgled grafičkog interfejsa za korisnika – početna stranica

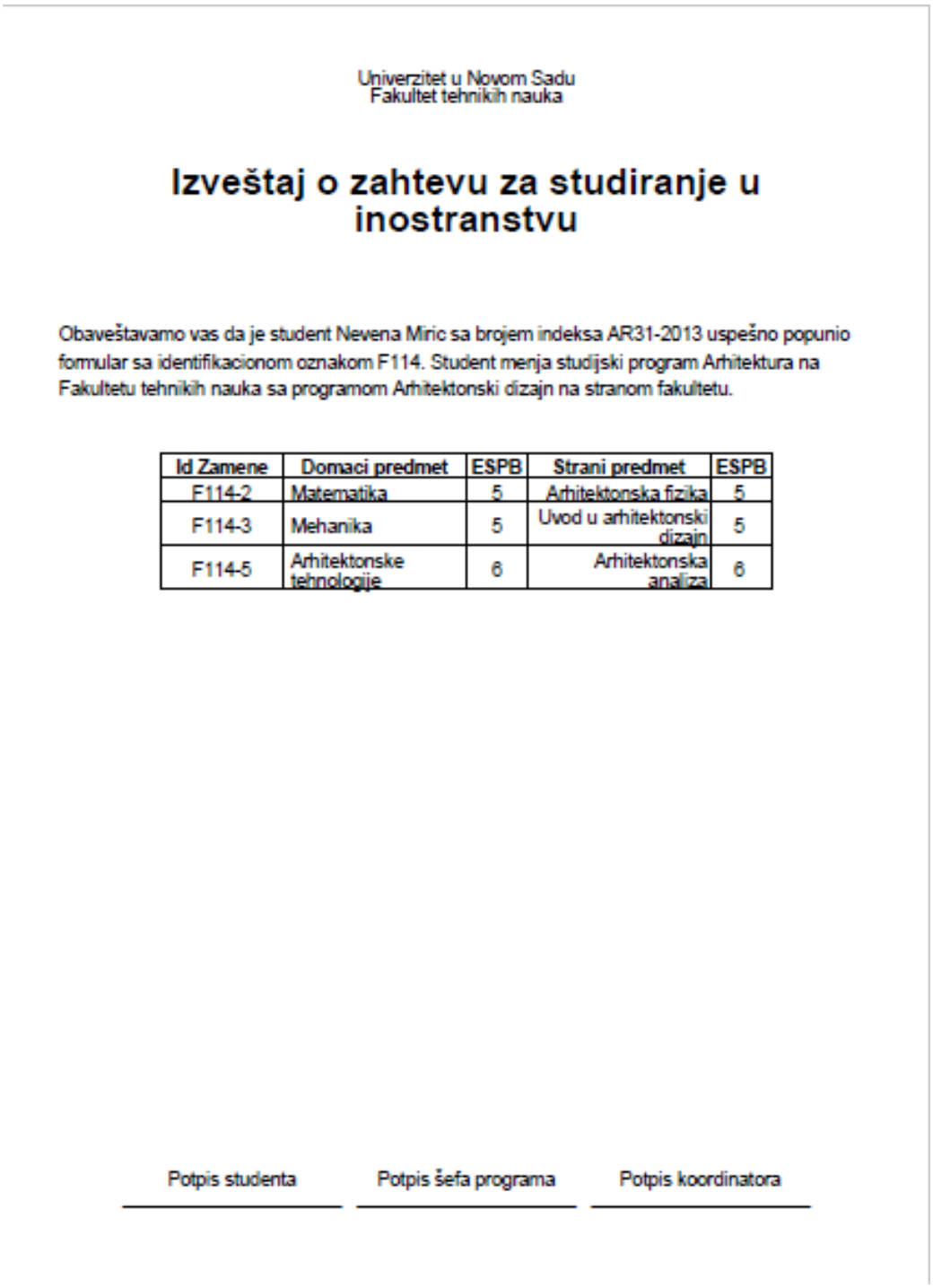


Slika 26 - Izgled grafičkog interfejsa za korisnika – prikaz formulara



Slika 27 - Izgled grafičkog interfejsa za nastavnika – odobrenje zamene

Slika 28 - Email za nastavnika



Univerzitet u Novom Sadu
Fakultet tehnikih nauka

Izveštaj o zahtevu za studiranje u inostranstvu

Obaveštavamo vas da je student Nevena Miric sa brojem indeksa AR31-2013 uspešno popunio formular sa identifikacionom oznakom F114. Student menja studijski program Arhitektura na Fakultetu tehnikih nauka sa programom Arhitektonski dizajn na stranom fakultetu.

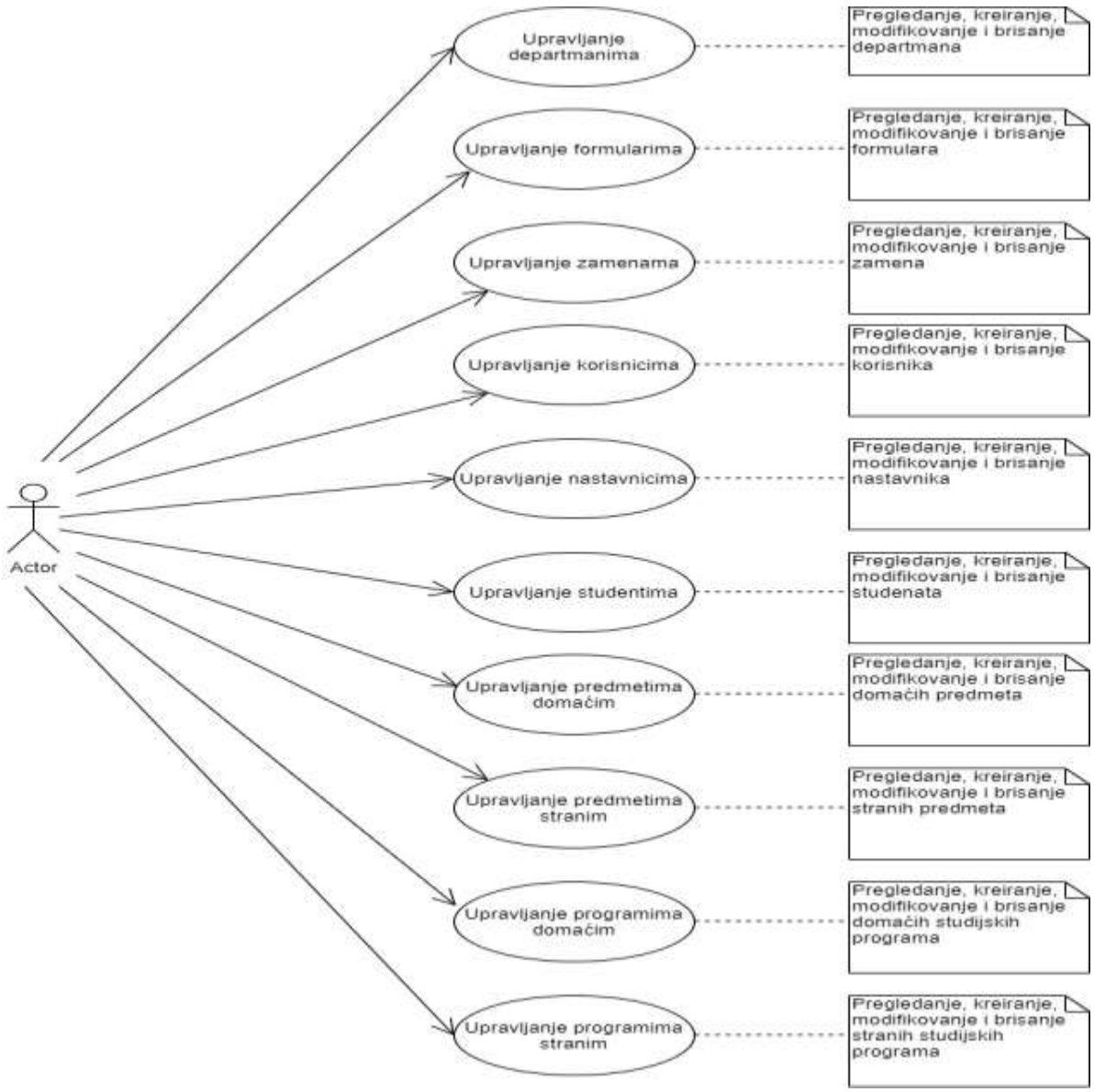
Id Zamene	Domaci predmet	ESPB	Strani predmet	ESPB
F114-2	Matematika	5	Arhitektonska fizika	5
F114-3	Mehanika	5	Uvod u arhitektonski dizajn	5
F114-5	Arhitektonske tehnologije	6	Arhitektonska analiza	6

Potpis studenta _____ Potpis šefa programa _____ Potpis koordinatora _____

Slika 29 - Izgled PDF-a koji se šalje studentu u prilogu

4.2.3 Administrator sistema

Administrator sistema ima pravo pregledanja, kreiranja, modifikovanja i brisanja svih entiteta u sistemu (slika 30). Postoje neka ograničenja, kao što je na primer, zabrana menjanja polja koja su primarni ključevi, ili brisanje entiteta koji u sklopu sebe imaju neke druge entitete – npr brisanje departmana koji u sklopu sebe ima domaće studijske programe, ili brisanje studijskih programa koji u sklopu sebe imaju studente koji na njima studiraju i predmete. Lančano brisanje je omogućeno samo u slučaju formulara, gde kada se obriše formular brišu se i njegove zamene, i brisanje studenata, gde se brišu svi studentovi formulari nakon što je student obrisan.



Slika 30 - Dijagram slučajeva korišćenja za administratora sistema

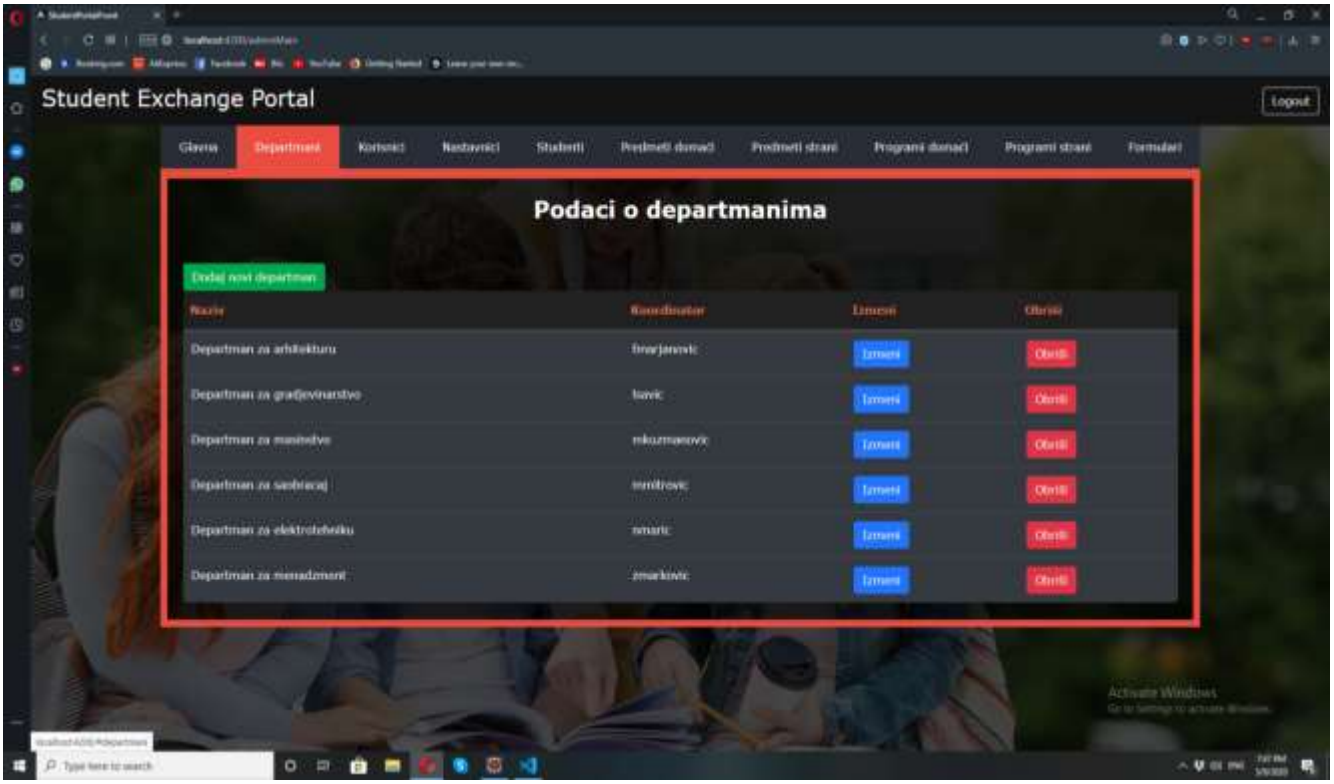
U kodu imamo 4 REST operacije:

- GET /departmani – ova operacija vraća listu svih objekata tipa departman klijentskoj strani, gde se oni tabelarno prikazuju
- POST /departmani – kreiranje novog departmana iz modalnog dijaloga sa klijentske strane. U telu REST zahteva se sa klijentske strane prosleđuju podaci o novom departmanu (u JSON formatu) koji će biti kreiran – naziv i koordinator departmana, pri čemu je koordinator tipa korisnik. Ukoliko je sve validno, novi departman se perzistira u bazi.
- PUT /departmani/{id} – izmena postojećeg departmana iz modalnog dijaloga sa klijentske strane. U telu REST zahteva se sa klijentske strane prosleđuju podaci o izmenjenom departmanu, pri čemu primarni ključ ne može da se menja (u ovom slučaju je to polje naziv). Ukoliko je sve validno, izmenjeni departman se perzistira u bazi.
- DELETE /departmani/{id} – brisanje departmana, koje je onemogućeno ako u okviru departmana postoje studijski programi – onemogućeno je lančano brisanje. Ako departman nema studijske programe, brisanje je moguće

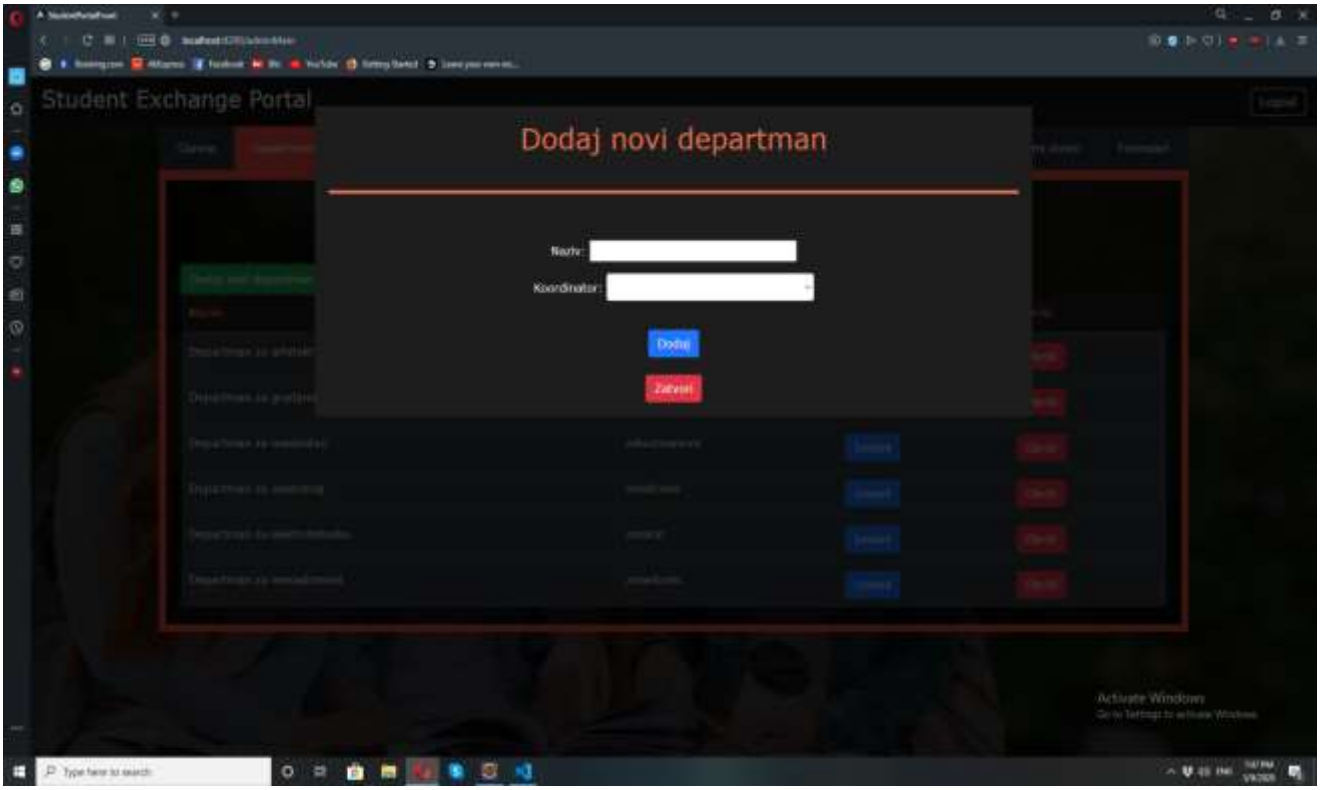
Implementacija je analogna za sve ostale entitete.

Poseban slučaj je dizajn interfejsa za formulare – ovde administrator može da kreira novi formular sa svim zamenama u njemu, da obriše postojeći formular, zatim da da odobrenje kao šef studijskog programa ili koordinator departmana, da dodaje, menja i briše zamene, ili da daje odobrenje u ime nastavnika za svaku zamenu.

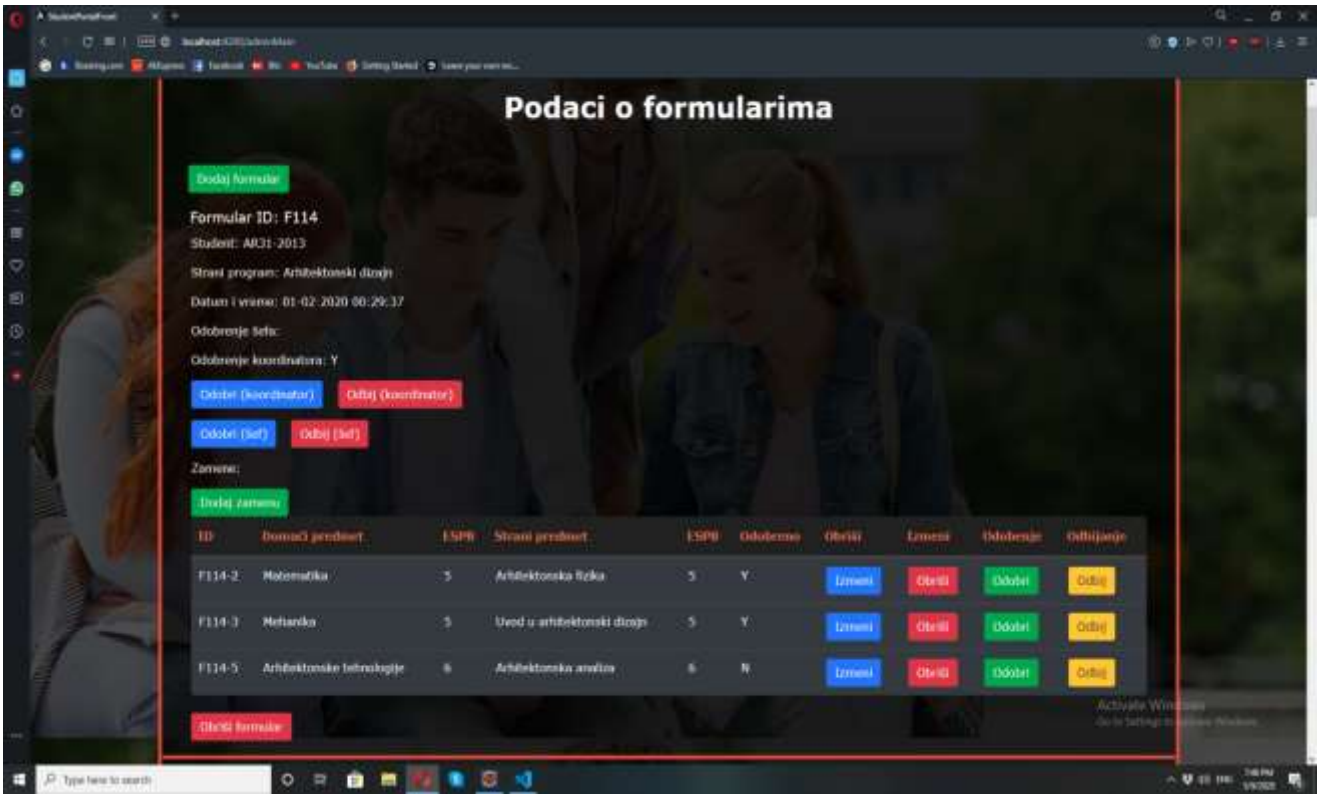
Primere izgleda grafičkog interfejsa za administratora vidimo na slikama 31, 32 i 33.



Slika 31 - Izgled grafičkog interfejsa za administratora na primeru departmana



Slika 32 - Izgled grafičkog interfejsa sa otvorenim modalnim dijalogom za dodavanje novog departmana



Slika 33 - Izgled grafičkog interfejsa za administratora na primeru formulara

5 Zaključak

U ovom radu opisan je istorijat sistema baziranih na znanju, kao i ekspertskih sistema. Date su smernice kada i kako se oni koriste, njihova struktura i način rada koji je ilustrovan na konkretnim primerima. Razmotrene su njihove prednosti i mane, zatim primeri programskih jezika i radnih okvira koji se koriste u sistemima baziranim na znanju, kao i detaljan teorijsko-praktični pregled na primeru programskog jezika Drools, uključujući i analizu Rete algoritma koji on koristi.

Jezici za pravila se često koriste u sinergiji sa web aplikacijama i zajedno se često koriste u rešavanju kompleksnih poslovnih problema. U ovom radu je opisana generalna arhitektura web aplikacija, kao i razlozi za njihovo korišćenje. Takođe, pokriveni su ciklus razvoja i održavanja web aplikacija i osnovna načela njihovog razvoja, koja je poželjno pratiti. Pošto su veb aplikacije izložene na internetu, a samim tim i potencijalnim napadima i malicioznom web saobraćaju, bilo je poželjno pokriti i oblast njihove bezbednosti, pa je i to urađeno.

Suština korišćenja pravila u poslovnim web aplikacijama je pokrivena kroz primer studentskog web servisa za studiranje u inostranstvu. Detaljno je objašnjena arhitektura kako klijentske tako i serverske strane, implementacija obe strane praćene ilustrativnim delovima koda, kao i objašnjenja funkcionalnih zahteva i uloga svakog korisnika aplikacije, propraćena dijagramima slučajeva korišćenja. Takođe su dati kratki opisi korišćenih tehnologija – Spring Boot-a i Angulara. Izbor tehnologije je uglavnom stvar preferenci programera, mada je potrebno uzeti u obzir mogućnosti i ograničenja tehnologija koje se koriste, jer nije svaka tehnologija podobna za svaki problem. Generalno, Angular i Spring Boot su skoro uvek dobri izbori zbog konstantnog održavanja i dodavanja novih funkcionalnosti i biblioteka, što ih čini jednim od najzastupljenijih tehnologija kada su u pitanju web aplikacije. Pored toga, lako se integrišu sa postojećim tehnologijama, uključujući i Drools alat. Na pitanje da li je Drools adekvatan (ili potreban) alat za problem ne postoji jednostavan odgovor, jer postoji mnogo faktora koje treba uzeti u obzir. Ovaj rad je nastojao da ukaže na prednosti i ograničenja korišćenja ovog alata, i generalno alata za pravila, kao i okolnosti u kojima je optimalno koristiti ovakve alate.

Jedan od sledećih koraka u istraživanju bi mogao biti proširenje baze pravila tako da se potpuno eliminiše sistem odobravanja od strane korisnika, gde bi pravila dolazila do zaključka da li su zamene predmeta smislene ili ne. Jedan od mogućih pristupa ovom rešenju jeste da se radi analiza imena predmeta (na primer, predmet Matematika 1 bi mogao da bude zamenjen jedino predmetima koji imaju veze sa matematikom – npr. Matematička analiza). Ovo bi podrazumevalo delimičnu implementaciju obrade prirodnog jezika, što nije jednostavan zadatak. Možemo otići i korak dalje i dati specifikaciju plana i programa svakog predmeta, pa proveravati u kojoj meri se planovi i programi svake zamene predmeta poklapaju, i na osnovu toga dolaziti do zaključka da li su zamene smislene ili ne.

6 Literatura

[1] Haocheng, Tan 2017 IOP Conf. Ser.: Mater. Sci. Eng. 242 012111

[2] Cabitza, F., & Dal Seno, B. (2005). DJess-A Knowledge-Sharing Middleware to Deploy Distributed Inference Systems. *International Journal of Computer and Information Engineering*. **2**: 66–69

[3] Jackson, Peter (1998), *Introduction To Expert Systems* (3 ed.), Addison Wesley, p. 2, ISBN 978-0-201-87686-4

[4] Anonymous. (2019, January 3). *What is a rule-based system? What is it not?* ThinkAutomation. https://www.thinkautomation.com/eli5/what-is-a-rule-based-system-what-is-it-not/

[5] Anonymous. (2017, February 25) *Rule Based Architecture of an Expert System.* BrainKart. https://www.brainkart.com/article/Rule-Based-Architecture-of-an-Expert-System_8931/

[6] Liu, H., Gegov, A. & Cocea, M. Rule-based systems: a granular computing perspective. *Granular Computing.* **1,** 259–274 (2016). https://doi.org/10.1007/s41066-016-0021-6

[7] Anonymous. (2013, April 17). *Forward Chaining and backward chaining in AI.* Javatpoint. https://www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai

[8] Anonymous. (2013, April 19). *Difference between backward chaining and forward chaining*. Javatpoint. https://www.javatpoint.com/difference-between-backward-chaining-and-forward-chaining

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 22.3: Depth-first search, pp. 540–549.

[10] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990]. "22.2 Breadth-first search". *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 531–539. ISBN 0-262-03293-7.

[11] Anonymous. (2006, August 21). *Why use a Rule Engine?* Japan JBUG. http://www.jbug.jp/trans/jboss-rules3.0.2/ja/html/ch01s02.html

[12] Rudolph, George (2008, July 10) *Some Guidelines For Deciding Whether To Use A Rules Engine*. Jess – Sandia National Laboratories https://jessrules.com/guidelines.shtml

[13] Abhishek, Ahlawat (2014, September 19). *Drools: When not to use a Rule Engine.* StudyTonight. https://www.studytonight.com/drools/not-use-rule-engine

[14] Anonymous (2015, November 11). *Drools – Introduction*. Tutorialspoint. https://www.tutorialspoint.com/drools/drools_introduction.htm

[15] Anonymous (2015, November 12). *Drools - Frequently Used Terms.* Tutorialspoint. https://www.tutorialspoint.com/drools/drools_frequently_used_terms.htm

[16] Charles, Forgy (1982). "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem". *Artificial Intelligence*. **19**: 17–37. doi:10.1016/0004-3702(82)90020-0

[17] Berlioz, Carole-Ann (2011, February 21). *Origin of the Rete Algorithm*. Sparkling Logic. https://www.sparklinglogic.com/rete-algorithm-demystified-part-1/

[18] Berlioz, Carole-Ann (2011, March 14). *How the Rete Algorithm Works*. Sparkling Logic. https://www.sparklinglogic.com/rete-algorithm-demystified-part-2/

[19] Berlioz, Carole-Ann (2012, May 4). *Evolution of the Rete Algorithm*. Sparkling Logic. https://www.sparklinglogic.com/rete-algorithm-demystified-part-3/

[20] Thinkwik (2017, September 1). *Web Apps Development: The Best Way to Grow an Enterprise*. Medium. https://medium.com/@thinkwik/web-apps-development-the-best-way-to-grow-an-enterprise-c8948982102d

[21] Spool, Jared (2007, October 22). *7 Critical Considerations for Designing Effective Applications*. UX Articles by UIE. https://articles.uie.com/designing_effective_apps/

[22] Mahipal, Nehra (2019, August 5). *Why Businesses Are Migrating to Web Applications?*. DEV Community. https://dev.to/decipherzonesoft/why-businesses-are-migrating-to-web-applications-1m0o

[23] Khodak, Elena (2019, November 4) *What it Takes to Develop Enterprise Web Application?* Romexsoft. https://www.romexsoft.com/blog/enterprise-application/

[24] Peter M. Mell, Timothy Grance (2011) „The NIST Definition of Cloud Computing". National Institute of Science and Technology

[25] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures*(Ph.D.). University of California, Irvine

[26] Anonymous. (2016, June 23). *Spring Boot JPA*. Javatpoint. https://www.javatpoint.com/spring-boot-jpa

[27] Mulders, Michiel (2019, September 16). *What is Spring Boot?* Stackify. https://stackify.com/what-is-spring-boot/

[28] Anonymous. *Introduction to services and dependency injection*. Angular.io. https://angular.io/guide/architecture-services

[29] Anonymous. *Introduction to components and templates*. Angular.io. https://angular.io/guide/architecture-components

[30] Herawan, Dwika (2020, March 4). *What is MySQL: MySQL Explained For Beginners*. Hostinger.com Web Hosting https://www.hostinger.com/tutorials/what-is-mysql

[31] Bodrov-Krukowski, Ilya (2018, March 22). *Angular Introduction: What It Is, and Why You Should Use It*. SitePoint. https://www.sitepoint.com/angular-introduction/