Rukovanje pravilima u poslovnim veb aplikacijama

# Uvod

# Sistemi bazirani na pravilima i poslovne veb aplikacije

## 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 formalno pojavljuju 1965. godine na univerzitetu Stanford, gde su istraživači, predvođeni Edvardom Fajgenbaumom, 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 npr dijagnostika bolesti (Mycin, 1972), identifikacija nepoznatih organskih molekula (Dendral, 1969), ili geološka analiza tla i minerala (Prospector, 1979). 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.

*Edvard Fajgenbaum*

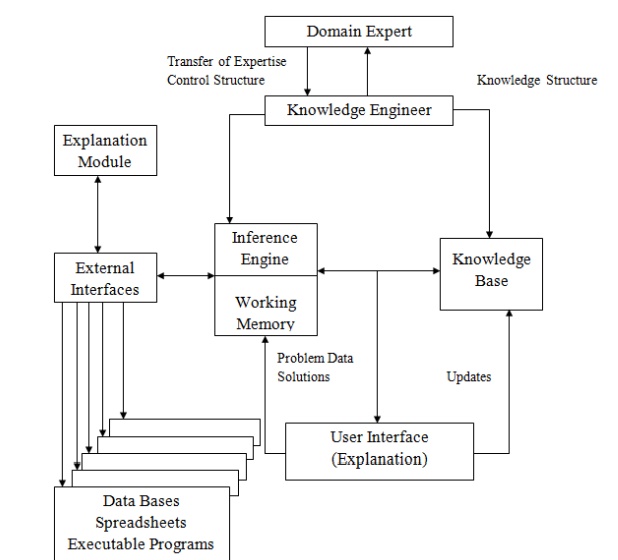
Moderni sistemi bazirani na pravilima i dalje primenjuju ideje ustanovljene od strane pionira, ali su često integrisani unutar velikih, kompleksnih sistemima. Ograničenja na koja su rani istraživači nailazili su ih naterala da potraže nove metode pristupa problemu, da bi se što vernije replicirao ljudski 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.

## Struktura sistema baziranih na pravilima

Tipičan sistem baziran na pravilima sadrži 4 komponente:

* 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, izvrši akciju
  + Cilj je da se automatizuju procesi, uz razbijanje tih istih procesa na korake
* Modul za zaključivanje (rezoner), koji vrši zaključivanje na osnovu ulaznih podataka i baze pravila. Zaključivanje se vrši u 3 koraka:
  + Podudaranje – provera da li su uslovi pravila zadovoljeni, nakon čega se kreira uređena lista elemenata radne memorije koji zadovoljavaju levu stranu pravila
  + Razrešenje konflikta, odnosno određivanje redosleda u kom će se desne strane pravila koja zadovoljavaju uslove izvršiti
  + Akcija, odnosno izvršenje desnih strana pravila u redosledu određenom tokom faze razrešenja konflikta
* 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
  + Kako – 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.
  + Zašto – modul za obrazlaganje mora da objasni zašto je određena informacija potrebna da bi se kompletirao korak u rezonovanju
* Privremenu radnu memoriju, koja sadrži činjenice (fakte)
* Korisnički interfejs ili neku drugu vrstu povezivanja sa spoljnim svetom, koja služi za slanje i primanje ulaznih/izlaznih signala i komunikaciju sa korisnikom



*Struktura sistema baziranog na pravilima*

## 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.

## Strategije zaključivanja

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

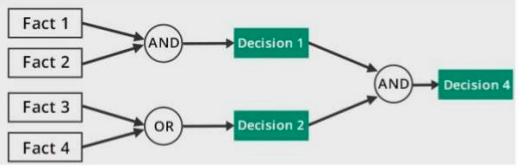
Kod ulančavanja unapred, počinje se od poznatih činjenica, i pomera se unapred primenom pravila na te činjenice, kako bismo došli do novih činjenica, i tako sve dok ne stignemo do cilja (zaključka). Zbog ovoga se ova strategija naziva i *zaključivanje vođeno podacima (data-driven),* ili *pristup* *odozdo 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*. 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. 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*. 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...





*Pretraga prvi u širinu Pretraga prvi u dubinu*



*Ulančavanje unapred*



*Ulančavanje unazad*

## Programski jezici bazirani na pravilima

* AWK – za procesuiranje teksta, ekstraksciju podataka i alat za izveštavanje
* CLIPS – najrasprostranjeniji alat za ekspertske sisteme, objektno orijentisan
* Constraint Handling Rules – deklarativni jezik namenjen rešavanju kombinatornih problema, takođe ima primenu u gramatičkoj indukciji, abduktivnom rezonovanju, multiagentskim sistemima, procesuiranju prirodnog jezika, prostorno-vremenskom rezonovanju, itd.
* Drools
* GOAL agent programming language – koristi se za programiranje kognitivnih agenata
* Jess – deklarativni jezik namenjen za Java platformu, nadskup CLIPS-a, ima primenu u automatizaciji ekspertskih sistema
* OPS5 – prvi jezik korišćen u uspešnom ekspertskom sistemu, za konfigurisanje VAX računara
* Prolog – 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 – programski sistem namenjen za decu, prezentovan u formi animiranih likova
* Mathematica – široko rasprostranjen moderan računarski sistem koji podržava neuronske mreže, mašinsko učenje, procesuiranje slika, geometriju, vizualizacije, itd.
* Wolfram – jezik koji podržava velik broj paradigmi, sa akcentom na funkcionalnom programiranju, simboličkom računanju i programiranje bazirano na pravilima

## Prednosti i mane sistema baziranih na pravilima

Neke od glavnih prednosti sistema baziranih na pravilima su sledeće:

* 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
* Razdvajanje 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
* Brzina i skalabilnost – sistemi bazirani na pravilima često su praćeni algoritmima koji efikasno uparuju podatke i pravila (npr Rete algoritam, Leaps algoritam, itd.)
* Centralizacija znanja – koristići pravila kreiramo repozitorijum znanja koji nam služi kao izvor istinitosti za poslovnu logiku – idealno su pravila toliko čitljiva da mogu da služe i kao dokumentacija
* Sposobnost objašnjavanja – pravila nas obaveštavaju o svim odlukama, kao i zašto su one donete
* Razumljivost – sintaksa pravila je bliža prirodnom jeziku, i samim tim je razumljivija ljudima koji nemaju tehničke sposobnosti programera
* Mogućnost da se menjaju pravila bez ponovnog startovanja aplikacije

Nedostaci sistema baziranih na pravilima:

* Alati koji barataju pravilima su često resursno zahtevni
* Narušavanje objektno orijentisanog koncepta na kom počivaju mnoge aplikacije, logika se 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
* 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
* “Eksplozija 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
* Sintaksa pravila se mora naučiti i to zahteva vreme – što programeri često nisu voljni da rade

Kada koristiti sisteme bazirane na pravilima?

* Kada ne postoji zadovoljavajuće tradicionalno programersko rešenje problema
* Kada je poslovna logika podložna čestim promenama
* Kada domenski eksperti nemaju adekvatno programersko znanje – lakše ih je podučiti sintaksi pravila nego sintaksi programskog jezika

Kada ne koristiti sisteme bazirane na pravilima?

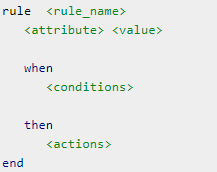
* Ukoliko je mali projekat, sa manje od 20 pravila, korišćenje pravila je nepotrebno
* Ukoliko je poslovna logika dobro definisana i statična, nepodložna promenama
* Ukoliko su pravila previše prosta, i uključuju rad nad malim brojem objekata
* Ukoliko je primarni fokus na performansu, u tom slučaju je optimalnije ostaviti logiku u samom kodu i ne koristiti pravila
* Ukoliko problem nije prikladan za pisanje u pravilima, ili je teško da se izrazi pravilima

## 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
* vreme izvršavanja (runtime), odnosno proces pravljenja objekata u radnoj memoriji i regulisanja aktivacije pravila

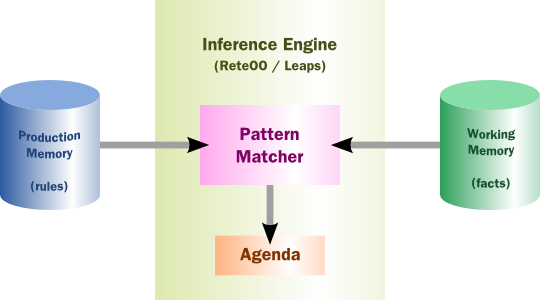
Kao i kod većine jezika baziranih na pravilima, pravila u Drools-u su u formatu if-then – ako je neki uslov zadovoljen, izvrši akciju.

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.

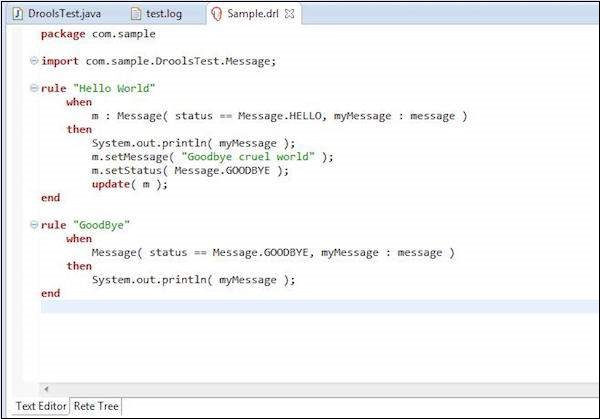
*Pseudokod jednog pravila*

Terminologija u Drools-u:

* Pravila – fajlovi sa ekstenzijom .drl u kojima navodimo uslove i akcije koje treba da se izvrše ako su uslovi zadovoljeni (when ‘a’ then ’b’)
* Činjenice – podaci nad kojima se pravila izvršavaju. U Javi, to su obično POJO objekti
* Sesija – glavna komponenta koja služi za okidanje pravila. Činjenice se ubacuju u sesiju i kada su uslovi ispunjeni, izvršavaju se desne strane pravila. Dva tipa sesije: stateless i stateful.
  + Stateless – stanje sesije između poziva se ne održava, za svaki poziv se pravi nova sesija
  + Stateful – pamti stanje između poziva
* Aktivacija – akcija, desna strana pravila
* Agenda – logički koncept, mesto na kom aktivacije čekaju da budu izvršene, odnosno mesto na kom se nalaze sve aktivacije čiji su uslovi zadovoljeni



*Struktura Drools-a*



*Primer jednog pravila u Drools-u*

## 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

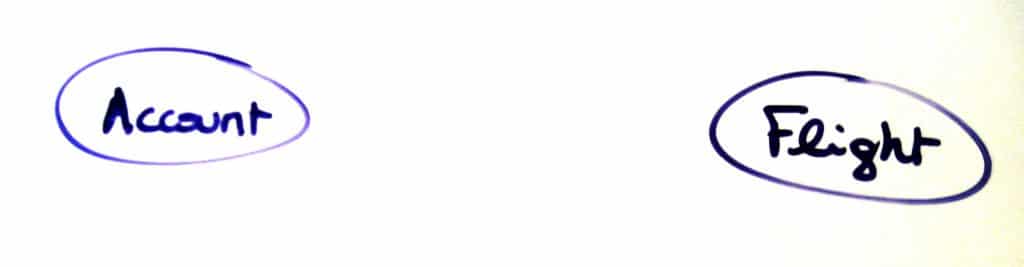
Ako je status Srebrni i u toku jednog meseca je imao više od 5 letova, dodeli bonus 1000km

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

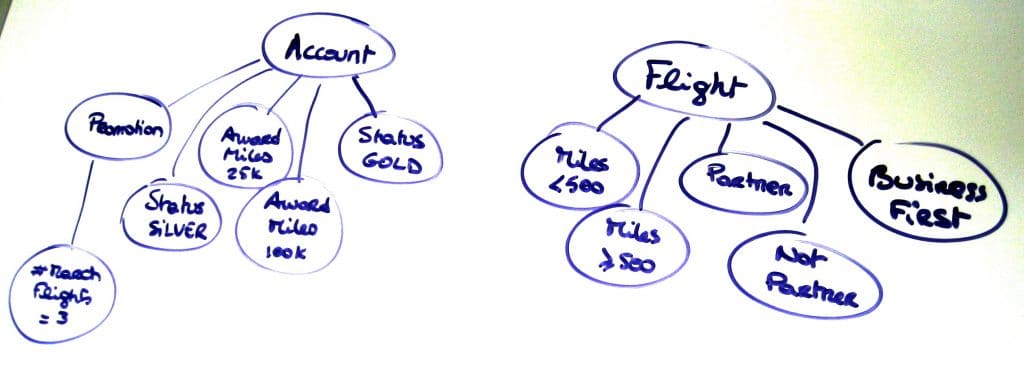
Ako je korisnik dobio status Zlatni, dodeli 8 sertifikata unapređenja

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.

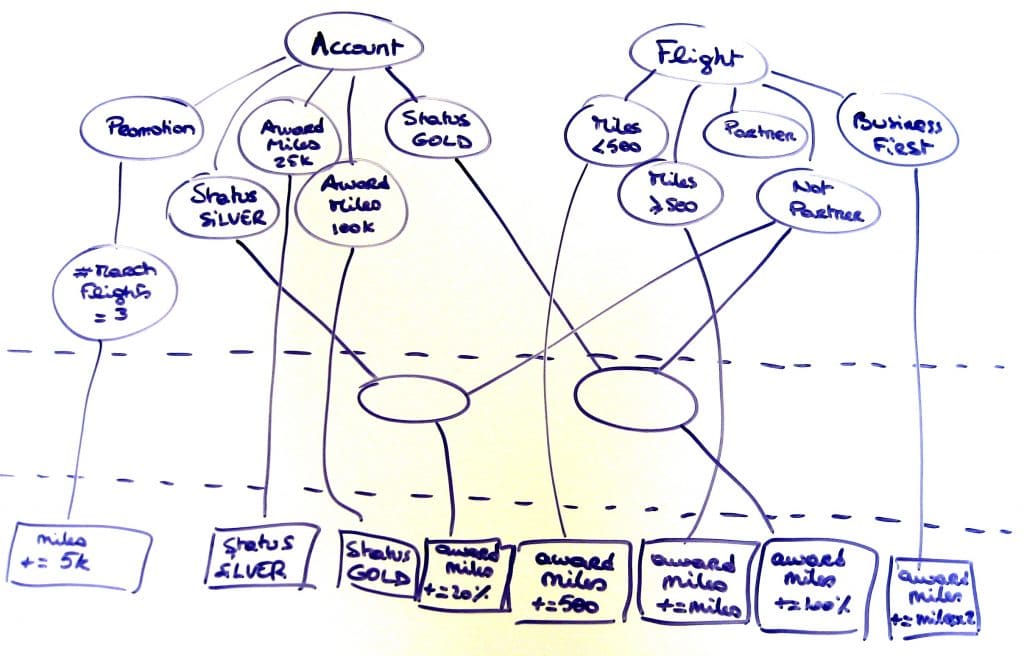
Rete mreža predstavlja osnovu algoritma. Sačinjena je od čvorova koji sadrže listu objekata koji zadovoljavaju određeni uslov. Diskriminantno stablo je prvi deo Rete mreže. Počinje sa *alfa čvorovima* koji se asociraju sa klasama (u objektno-orijentisanom smislu). Sve instance jedne klase će biti elementi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



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.



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.



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.

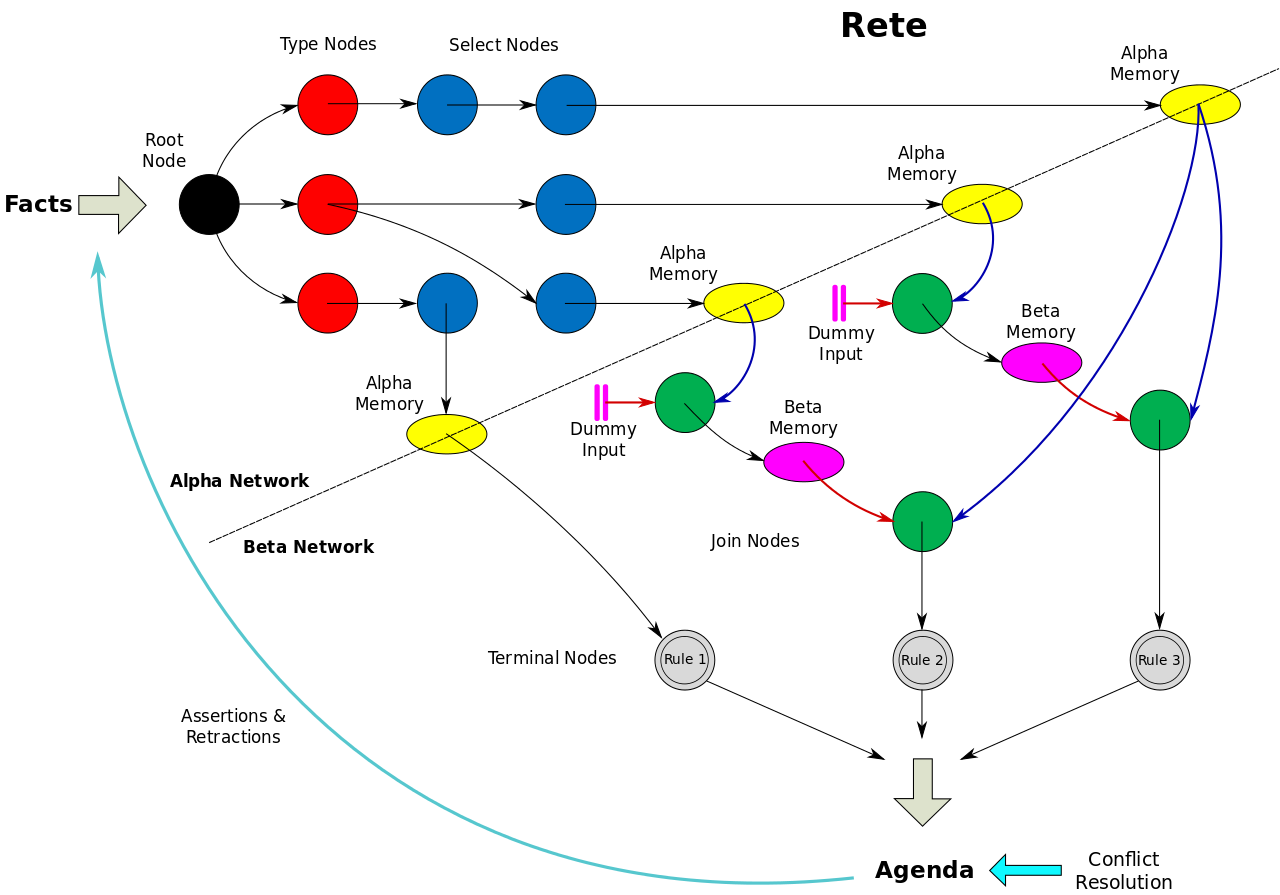
Pun primer: Nikola leti iz Beograda za Stokholm – kilometraža leta je 2300 km. Nikola ima ukupnu kilometražu od 23400 km. Nikola ne ispunjava uslove za srebrnog ili zlatnog člana. Let ima više od 500km, pa ćemo mu dodeliti tu vrednost leta. Nikola leti ekonomskom klasom, pa nema bonuse za biznis/prvu 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.

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 i OPSJ-u.

Rete III – razvijen početkom 2000-ih, razvijen u saradnji sa FICO inženjerima, u suštini predstavlja Rete II algoritam implementiran kao deo FICO Advisor engine-a.

Rete-NT – razvijen 2010, oko 500 puta brži od originalnog algoritma, i 10 puta brži od svog prethodnika, Rete II.



*Grafički prikaz Rete algoritma*

## 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.



*Arhitektura web aplikacije*

Web aplikacije prate sledeći lanac komandi:

* Korisnik upućuje zahtev preko interfejsa na pretraživaču (tzv. *frontend*), koji se prosleđuje ka serverskoj mašini
* Server prosleđuje zahtev odgovarajućoj web aplikaciji (tzv. *backend*)
* Web aplikacija izvršava zadatak i generiše izveštaj
* Web aplikacija šalje odgovor serveru
* 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
* Poboljšana bezbednost – većina web aplikacija je instalirama na namenskim serverskim mašinama, koje održavaju eksperti u oblasti administracije i bezbednosti. Ovo čini obezbeđivanje aplikacije jeftinim i pouzdanim
* Bolji performans – moderni web serveri obično čuvaju web aplikacije na SSD-u i pokreću ih pomoću moćnih CPU-ova i GPU-ova. U pogledu performansa, desktop aplikacije ne mogu da se mere sa web aplikacijama kada su u pitanju procesi koji zahtevaju mnogo hardverskih resursa
* Raspoloživost 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 – 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
* Fleksibilnost – pristup poslovnim platformama je lak bez obzira na to gde se korisnik nalazi, potrebno je samo da ima pristup internetu
* Bolja korisnička podrška – za razliku od desktop aplikacija gde se podrškom rukuje zasebno, kod web aplikacija nudi ugrađenu korisničku podršku kojoj korisnici mogu da pristupe onlajn
* 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).

## 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 i brze
* Razumljive su 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,...
* *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:

* Definisanje strategije – dokumentovanje svrhe aplikacije, ciljevi i smer razvoja
* Ispitivanje budućih korisnika aplikacije
* Kreacija funkcionalnog dizajna aplikacije
* Izbor tehnologije, bazira se na specifikaciji projekta
* Dizajniranje vizuelnog izgleda aplikacije
* Definisanje strukture baze podataka
* Kodiranje
* Testiranje, dokaz o kvalitetu, pregledanje i isporuka
* Održavanje i podrška

U fazi razvoja poslovnih aplikacija, bitno je spomenuti i cloud servise. Cloud servisi predstavljaju isporuku računarskih resursa i skladišnih kapacitetakao 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.

## 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.

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.

## 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.

Neke smernice za osiguravanje web aplikacija:

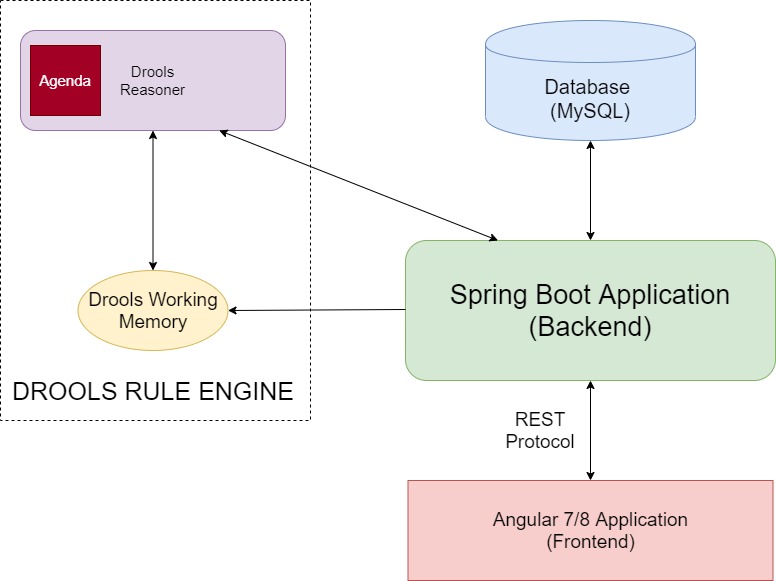
* Rigorozno testiranje ulaznih parametara – ulazni parametri su potencijalan put do osetljivih podataka i internih sistema. Ako softver povlači podatke sa Interneta, neko će eventualno pokušati da zaobiđe bezbednosne mere i prouzrokuje problem. Dobar primer za ovo je SQL injekcija. Na primer, na formi 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.
* Autentikacione metode bez lozinke – lozinke nisu toliko bezbedan mehanizam za čuvanje bitnih podataka, treba potražiti alternative, kao što su N-faktor autentikacija, SAML ili FIDO protokol.
* 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.

# Arhitektura sistema

## Struktura sistema

Studentski portal za mobilne studente je web aplikacija pisana u Javi, odnosno Javinom frameworku Spring Boot na backend-u i Angularu 8 na frontend-u i služi za kreiranje formulara od strane studenata sa listom predmeta na domaćem fakultetu koje menja sa predmetima koje će slušati na stranom fakultetu. Komponente korisničkog interfejsa komuniciraju sa backend-om 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.



# Implementacija

# Zaključak

# Literatura

<https://en.wikipedia.org/wiki/Rule-based_system>

<https://en.wikipedia.org/wiki/Expert_system>

<https://www.thinkautomation.com/eli5/what-is-a-rule-based-system-what-is-it-not/>

<http://www.cs.stir.ac.uk/courses/ITNP60/lectures/2%20Decision%20Support/1%20-%20Rule%20Based%20Systems.pdf>

<https://www.brainkart.com/article/Rule-Based-Architecture-of-an-Expert-System_8931/>

<https://link.springer.com/article/10.1007/s41066-016-0021-6>

<https://www.javatpoint.com/forward-chaining-and-backward-chaining-in-ai>

<https://www.javatpoint.com/difference-between-backward-chaining-and-forward-chaining>

<https://en.wikipedia.org/wiki/Depth-first_search>

https://en.wikipedia.org/wiki/Breadth-first\_search

<https://en.wikipedia.org/wiki/List_of_programming_languages_by_type#Rule-based_languages>

<http://www.jbug.jp/trans/jboss-rules3.0.2/ja/html/ch01s02.html>

<https://stackoverflow.com/questions/250403/rules-engine-pros-and-cons>

<https://jessrules.com/guidelines.shtml>

<https://www.studytonight.com/drools/not-use-rule-engine>

<https://www.tutorialspoint.com/drools/drools_introduction.htm>

<https://www.tutorialspoint.com/drools/drools_frequently_used_terms.htm>

<https://en.wikipedia.org/wiki/Rete_algorithm>

<https://www.sparklinglogic.com/rete-algorithm-demystified-part-1/>

<https://www.sparklinglogic.com/rete-algorithm-demystified-part-2/>

<https://www.sparklinglogic.com/rete-algorithm-demystified-part-3/>

<https://medium.com/@thinkwik/web-apps-development-the-best-way-to-grow-an-enterprise-c8948982102d>

<https://articles.uie.com/designing_effective_apps/>

<https://articles.uie.com/designing_effective_apps_part2/>

<https://dev.to/decipherzonesoft/why-businesses-are-migrating-to-web-applications-1m0o>

<https://www.romexsoft.com/blog/enterprise-application/>

https://www.sciencedirect.com/topics/computer-science/enterprise-web-application

<https://sr.wikipedia.org/wiki/Računarstvo_u_oblaku#Ključne_karakteristike_računarstva_u_oblaku>

<https://en.wikipedia.org/wiki/Representational_state_transfer>