Prolog i Semantički Web

Kišić, Sara

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:143540

Rights / Prava: Attribution-ShareAlike 3.0 Unported/Imenovanje-Dijeli pod istim uvjetima 3.0

Download date / Datum preuzimanja: **2025-03-20**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Sara Kišić

PROLOG I SEMANTIČKI WEB

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Sara Kišić

Matični broj: 0016123012

Studij: Baze podataka i baze znanja

PROLOG I SEMANTIČKI WEB

DIPLOMSKI RAD

Mentorica:

Prof. dr. sc. Sandra Lovrenčić

Varaždin, rujan 2021.

Sara Kišić

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Diplomski rad daje uvid u područje semantičkog weba, njegovih tehnologija i primjene, te opisuje programski jezik Prolog koji osim što ima veliku primjenu u logičkom programiranju, odlično surađuje s tehnologijama semantičkog weba. Kao praktični dio, napravljena je aplikacija koja prikazuje primjenu semantičkog weba korištenjem programskog jezika Prolog.

Ključne riječi: prolog; semantički web; logičko programiranje;

Sadržaj

1. Uvod

Semantički Web započeo je kao ambiciozan projekt 2000.-ih godina. Za razliku od tadašnjeg (i trenutnog) Web-a koji je bio čitljiv isključivo ljudima, semantički Web bi bio čitljiv i od strane računala, što daje veliki niz novih mogućnosti. Tim Berners – Lee prvi je opisao svoju viziju novog Web-a koji bi omogućio ljudima i računalima potpunu kooperaciju i tako delegirao neke jednostavne zadatke na računala i time omogućio automatizaciju svakodnevnih zadataka.

Diplomski rad daje uvid u same početke semantičkog Web-a, tehnologije koje su se razvile, probleme koji su nastali prilikom razvoja, ali i današnje stanje semantičkog Web-a. Spominje se i programski jezik Prolog za kojeg su razvijeni određeni paketi kako bi se omogućio rad na aplikacijama semantičkog Web-a, te su opisani njegovi osnovni koncepti koji su potkrijepljeni kratkim primjerima kako bi čitatelj spojio teoriju s praktičnim dijelom, te što bolje svladao same koncepte logičkog programiranja.

Praktični dio diplomskog rada daje bolji uvid u tehnologije semantičkog Web-a, a posebno ontologije, SPARQL upite i RDF. Prikazan je jedan od načina kako su tehnologije semantičkog Web-a spojene s Prolog programskim jezikom te ostalim jezicima za kreiranje Web aplikacije. Opisani su dostupni programski paketi, njihove prednosti i mane u korištenju te je prikazan sam rad aplikacije.

2. Semantički Web

Semantički web se definira kao „ekstenzija trenutnog Web-a u kojem je informacijama pridodano detaljno definirano značenje koje omogućuje kooperaciju ljudi i računala“ [7]. Kako je to ekstenzija trenutnog Web-a, potrebno je prije razrade teme definirati i sam Web te se pobliže upoznati s njegovim dijelovima, poviješću samog Web-a, njegovim prednostima i nedostacima.

2.1. Web

World Wide Web ili popularno zvan samo Web je kolekcija dokumenata koji se nazivaju web stranice i nalaze se na Internetu [1]. Često se pogrešno naziva i Internet, no bitno ih je razlikovati. Kao što je spomenuto, Web je kolekcija dokumenata koji su međusobno povezani putem hiperlinkova (riječi, rečenica ili slika koje korisnik može kliknuti kako bi ga odvele na drugi odlomak ili stranicu istog ili drugog dokumenta) [2]. S druge strane, Internet je golema globalna mreža koja povezuje milijarde različitih računala i ostalih elektroničkih uređaja [3].

Godine 1989. Tim Berners – Lee je pokušao pronaći način kojim bi znanstvenici međusobno dijelili informacije i saznanja svojih eksperimenata. U to je vrijeme koncept Interneta i hiperteksta već bio poznat, no nitko se još nije sjetio povezati sve dostupne dokumente. Tada je Tim Berners – Lee predložio korištenje tri glavne tehnologije koje sačinjavaju Web: HTML, URL i HTTP. One ujedno čine i arhitekturu Web-a. Ukratko, HTML (engl. *Hyper Text Markup* Language) je standardni jezik za kreiranje Web stranica, a sastoji se od raznih elemenata koji govore pregledniku na koji način prikazati sadržaj nekog Web mjesta [4]. URL (engl. *Uniform Resource Locator*) je adresa koja je dana nekom resursu na Webu. Svaka URL adresa vodi na neko jedinstveno Web mjesto ili resurs poput slike, videa i slično. URL je zapravo identifikator nekog resursa koji se nalazi na Web-u koji služi i za opis same lokacije na kojoj se taj resurs nalazi [5]. Treća korištena tehnologija je HTTP protokol koji pripada aplikacijskom sloju protokola i služi za prenošenje hipermedijskih dokumenata kao što su HTML stranice. Njegova primarna svrha je uspostava komunikacije između web preglednika (klijenta) i web servera. Ova komunikacija se vrši preko mrežnog sloja (protokol TCP – IP) [6].

Također, napravio je i prvi preglednik (engl. *browser*) 1991. godine. Ovime je Berners – Lee otvorio mogućnost korištenja Interneta i njihovih sadržaja svijetu [1].

S vremenom je Web prilično evoluirao, a u sljedećih nekoliko odlomaka ukratko je opisana njegova evolucija počevši od njegove prve pojave (Web 1.0) pa sve do danas (Web 3.0).

Prva verzija Web-a, Web 1.0 (drugi naziv je i sintaktički web (engl. *syntactic web*) ili samo za čitanje (engl. *read-only Web*)) odvijala se u eri od 1990. godine pa do 2000. godine. Korisnici Web-a mogli su samo čitati informacije koje su im tada bile dostupne te nisu imali mogućnost da sami kreiraju svoj sadržaj ili da komuniciraju s kreatorom sadržaja. Ovaj Web sastojao se samo od statičnih web stranica [6]. Tvrtke koje su u to vrijeme posjedovale svoju web stranicu, korisnicima su mogle pružiti isključivo svoj katalog ili neku brošuru što je vrlo sličilo tadašnjim reklamama koje su se mogle naći u novinama. Ove stranice nisu često bile ažurirane i kao što je spomenuto, bile su poprilično statične, što znači da osim čitanja, korisnik nije bio u nikakvoj drugoj interakciji sa samom stranicom ni tvrtkom koja tu stranicu posjeduje [8].

Web 2.0 se još naziva i društveni Web (engl. *Social Web*) ili čitaj-piši Web (engl. *read – write Web*). Njegova era počinje 2000.-ih godina i njegov razvoj traje sve do 2010.-ih godina, te je u upotrebi još i danas. Primarna karakteristika ove verzije Web-a je mogućnost komunikacije između korisnika Web-a i stranica koje posjećuju. Za razliku od prethodne verzije, ovdje svaki korisnik Web-a može kreirati i sadržaj koji se na njemu prikazuje, tj. svaki korisnik može kreirati vlastitu stranicu i popuniti je sadržajem kojim želi [6]. U ovom razdoblju pojavljuju se i nove tehnologije poput blogova, RSS-a (engl. *Really Simple Syndication* – informira korisnike o novim ažuriranjima blogova ili web stranica koje prate), Wiki stranica i ostalog [8]. Dolazi i do pojave društvenih mreža poput Facebook-a, Youtube-a, Twitter-a… Također, koriste se web tehnologije poput HTML5, CSS3, Javascript okvira, AJAX i ostali [6].

Nakon 2010-ih, a neki tvrde čak i prije, pojavljuje se nova verzija Web-a koju je Tim Berners – Lee još nazvao i semantičkim Web-om ili čitaj – piši – izvrši Web (engl. *read – write – execute Web*). Ova verzija Web-a fokusira se na sve ono što bi Web mogao postati, dakle na budućnost samog Web-a i njegove upotrebe. Računala pomoću umjetne inteligencije i strojnog učenja mogu interpretirati informacije na sličan način kao i ljudi. Ovime je korisnicima omogućeno brže pronaći točno ono što traže, prepoznati ljude, mjesta, razne događaje, proizvode i ostalo [6]. U suštini, podaci na Web-u će biti međusobno povezani na decentralizirani način što se smatra velikim korakom naprijed u odnosu na trenutnu generaciju Web-a (Web 2.0.) gdje su podaci uglavnom pohranjeni na centraliziranim repozitorijima. Cilj semantičkog Web-a je omogućiti korisnicima i računalima interakciju s podacima koji se nalaze na Web-u, a da bi se to ostvarilo programi trebaju podatke interpretirati konceptualno i kontekstualno, dakle, trebali bi shvatiti koncept samog podatka kojeg žele protumačiti, ali i kontekst u kojem se taj podatak primjenjuje. Dakle, sama svrha

Web-a 3.0. je pružanje osobnijeg iskustva pretraživanja Web-a te još veća integracija interneta u svakodnevni život čovjeka kako bi se pojednostavili svakodnevni zadaci [9].

Web sadrži zapanjujuće mnogo informacija, od vremenske prognoze pa sve do članaka o najnovijim znanstvenim otkrićima. Pristup tim informacijama je prilično jednostavan, ako korisnik zna koje ključne riječi koristiti kako bi pristupio onome što traži. Kao što je definirano, semantički Web se naziva ekstenzijom trenutnog Web-a te on pokušava riješiti sljedeća dva problema: limitirani pristup podacima i nedostatak delegacije.

Problem limitiranog pristupa podacima se očitava u tome što su dokumenti indeksirani i pristupa im se putem običnog teksta. Za pretraživanje tih dokumenata koristi se običan algoritam zasnovan na podudaranju nizova tekstualnih znakova (engl. *strings*). Ovo predstavlja problem za riječi koje mogu biti dvosmislene. Na primjer, ako korisnik pretražuje riječ „Paris“, ona može predstavljati glavni grad Francuske, ime izmišljenih likova, naziv filmova, pjesama i ostalo. U tom slučaju korisnik sam treba filtrirati informacije koje smatra bitnima. Osim dvosmislenosti riječi, postoji i problem kod postavljanja kompleksnih pitanja. Npr. Google tražilica može odgovoriti na velik broj pitanja poput „Što je Web?“, „Tko je Tim Berners – Lee?“ itd. No, tražilica nema odgovor na pitanja o željama samog korisnika, njegovom karakteru, vrijednostima i ostalome. Primjer jednog kompleksnijeg upita bio bi i „Gdje ići na odmor idući vikend za manje od 1000€“. Iako tražilica neće dati konkretan odgovor na ovo pitanje, jer takva tehnologija još ne postoji, ponudit će velik broj web stranica i članaka koji sadržavaju ključne riječi koje se nalaze u pitanju, a daljnje istraživanje i samu rezervaciju će korisnik morati obaviti sam [12].

Drugi problem koji se spominje je nedostatak delegacije. Web je zapravo kolekcija statičkih dokumenata. Prilikom pretraživanja Web-a, računalo korisniku samo prikazuje podatke, a korisnik sam mora zaključiti što će s tim podacima napraviti i kako će ih koristiti. Svake godine nastane velika količina novih podataka na Web-u [12]. Na početku 2020. godine dan je podatak da se na Web-u trenutno nalazi oko 44 zetabajta podataka, što je nepojmljivo velika količina [13]. Tu dolazi do potrebe za delegacijom. Računala, tj. agenti bi sami integrirali informacije, analizirali pronađene podatke i davali vlastite zaključke. Time bi se maksimalno iskoristile informacije koje su dostupne na Web-u i tako olakšao posao korisnicima Web-a, što je i jedna od ideja semantičkog Web-a [12].

2.2. Uvod u semantički Web

Kao što je spomenuto u prethodnom odlomku, semantički Web je naziv za novu verziju Web-a koja se ponaša kao ekstenzija trenutnom Web-u. U svom radu, Tim Berners – Lee kao definiciju semantičkog Web-a navodi sljedeće: semantički Web informacijama daje

dobro definirano značenje koje omogućava ljudima i računalima međusobnu kooperaciju. On zamišlja okružje u kojem agenti sami pretražuju Web kako bi izvršili zadatke koje im je korisnik zadao. Ovakvo okružje ne bi upotrebljavalo naprednu umjetnu inteligenciju, već bi svaka stranica trebala imati ukodiranu semantiku pomoću dostupnog softvera za obavljanje ovakvih zadataka. Dakle, Tim Berners – Lee ne smatra da bi svatko trebao naučiti kako semantički Web i njegove tehnologije funkcioniraju, već želi omogućiti upotrebu softvera koji automatiziraju taj postupak i na temelju postavki koje vlasnik stranice odabere, dodaje odabrano semantičko značenje Web stranici [7].

Ciljevi semantičkog Web-a su uspostava strukture sadržaja koji se može pronaći na Web-u, te kreiranje okruženja u kojem bi agenti prolazeći po stranicama na Web-u sami izvršavali zadatke koji im korisnik zadaje. Jedno od bitnih svojstava koje bi omogućilo izvršavanje ovakvih scenarija je univerzalnost, a ono se u ovom slučaju ostvaruje upotrebom hiperveza [7].

Da bi semantički Web bio ostvariv, potrebno je najprije napraviti neku vrstu reprezentacije ili prikaza znanja (engl. *Knowledge Representation (KR)*). Prikaz znanja odnosi se na izgradnju modela svijeta, okružja, domene ili problema s kojim se korisnik suočava. Model se gradi na način da se razvija skup formalizama i pravila koja mogu pružiti visoku razinu opisa tog modela. Ti modeli se još nazivaju i ontologije, a o njima se detaljnije raspravlja kasnije u radu [10]. Pravila su potrebna kako bi sustav znao odgovoriti na pitanje ili problem koji mu je postavio korisnik. Ukoliko je pitanje presloženo i računalo ga ne razumije, ono neće točno odgovoriti na njega ili neće biti u mogućnosti odgovoriti uopće. Na tom principu radi mnogo sustava, no svaki od njih ima vlastita pravila koja se ne mogu dalje prenositi, što zapravo znači da isto pitanje može proizvesti nijedan, dva ili više različita odgovora u različitim sustavima. Jedan od zadataka semantičkog Web-a je stvoriti jezik koji opisuje podatke na samom Webu, ali i pravila za zaključivanje o podacima, te mogućnost izvoza tih pravila na Web kako bi ona bila primjenjiva u što više sustava [7].

Ukratko, Tim Berners – Lee govori o tome kako želi u sam Web dodati logiku koja upotrebljava pravila koja stvaraju zaključke, odabiru način djelovanja i odgovaraju na pitanja. Jedan od problema je taj da logika mora biti dovoljno kompleksna da se pomoću pravila mogu izraziti i kompleksni podaci, ali ne i toliko kompleksna da se tom logikom pokušava riješiti paradoks [7].

Semantički Web informacijama koje se nalaze na Web-u želi dodati formalno značenje koje daje jasniji opis same informacije. Time bi računala točnije mogla razlikovati informacije koje imaju isti naziv ili su slične što bi poboljšalo točnost donesenih zaključaka, pretraživanje i integraciju informacija [42]. Tehnologije kojima započinje razvoj semantičkog

Web-a već su postojale i to su XML (engl. *eXtensible Markup Language*) te RDF (engl. *Resource Description Framework*) [7]. Poglavlja koja slijede objašnjavaju tehnologije koje semantički Web upotrebljava.

2.3. Tehnologije semantičkog Web-a

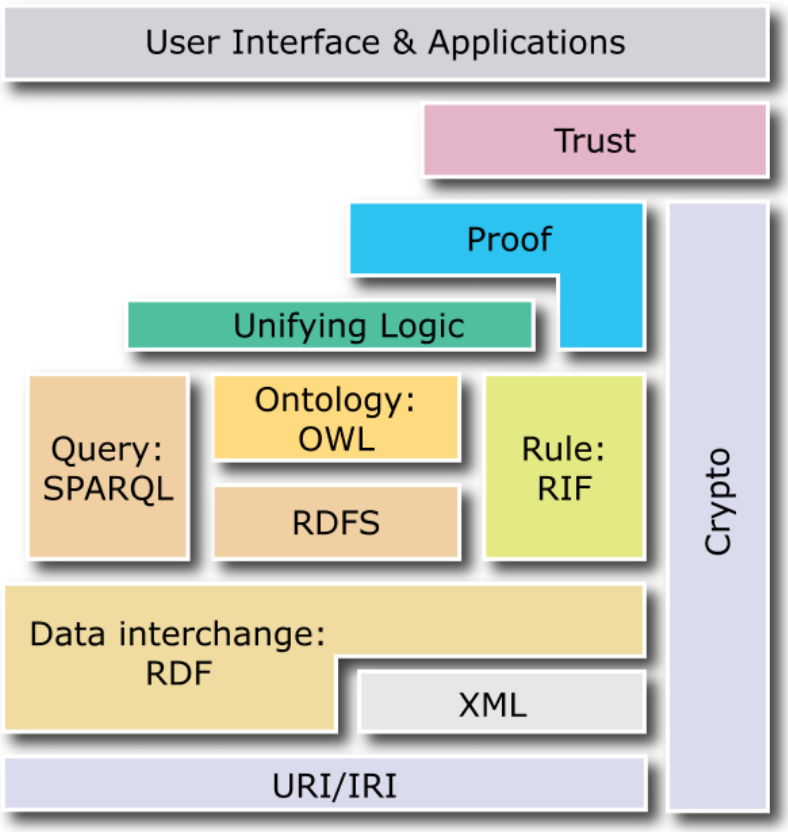
Da bi sadržaj Web-a bio razumljiv kako ljudima, tako i računalima, nije dovoljno pohraniti podatke Web stranica u obliku sintakse koja je čitljiva računalima. Na primjer, svaka HTML stranica je u nekoj mjeri čitljiva računalima, no nisu svi podaci koji se nalaze na toj stranici prikazani tom sintaksom iz koje bi računalo samostalno moglo donositi neke zaključke i interpretirati dane podatke. Semantički Web nastoji pristupiti što većem broju informacija i podataka, te smanjiti vrijeme koje je potrebno ljudima da te informacije i podatke sami pronađu [11].

Tehnologije semantičkog Web-a fokusiraju se na dodavanje semantike sadržajima Web-a. To uključuje jezike za označavanje (engl. *markup*) kao što su HTML ili XML. Pomoću HTML-a može se na razne načine izraziti semantika podataka. Najčešći način je pomoću META oznake. To je jedan od direktnih načina uključivanja semantike u dokumente na Web-u. Postoji i indirektni način, npr. pomoću H1 oznake (oznaka koja se tipično koristi za naslove odlomaka). Iz te oznake može se zaključiti da su informacije koje ta oznaka sadržava bitne i da opisuju sadržaj dokumenta koji se pregledava. Što se tiče XML-a, on je razvijen kao jezik za strukturiranje dokumenata na Web-u. On generalizira HTML na način da dopušta korisniku da sam definira svoje oznake, no baš zbog te fleksibilnosti smanjuju se šanse za točnu interpretaciju semantike podataka [12].

U tehnologije semantičkog Web-a želi se uključiti i upotreba tzv. inteligentnih agenata koji mogu razmjenjivati stečeno znanje i procesirati informacije kako bi asistirali korisnicima u svakodnevnim zadacima. Da bi upotreba tih agenata bila moguća, potrebno je koristiti i umjetnu inteligenciju (engl. *Artificial Intelligence, AI*). Osim toga, W3C (engl. *World Wide Web Consortium*) je objavio i razne standarde semantičkog Web-a koji olakšavaju razmjenu semantički bogatih informacija. Neki od tih standarda su RDF (engl. *Resource Description Framework*), OWL (engl. *Web Ontology Language*), SPARQL (engl. *Protocol and RDF Query Language*) i ostali [11].

2.3.1. Slojevi semantičkog Web-a

Tehnologije semantičkog Web-a mogu se prikazati sljedećom slikom (Slika 1).



Slika 1. Slojevi semantičkog Web-a [14]

Slika 1 prikazuje semantički Web kao skup jezika koji se nadograđuju na postojeće standarde kao što su XML, URI ili IRI. Mogu se još nazvati i komponentama semantičkog Web-a. Ovaj prikaz koji se sastoji od slojeva još se naziva i *The Semantic Web Layer Cake*, a koncept je osmislio i predstavio Tim Berners – Lee.

Model na kojem se zasniva ovaj prikaz naziva se OSI model (engl. *The Open Systems Interconnection model*). To je jedan od načina za podjelu komunikacijskih sustava na manje dijelove koji se nazivaju slojevima. Slojevi se sastoje od konceptualno sličnih funkcija koje pružaju usluge slojevima iznad, a dobivaju usluge od slojeva ispod [12].

Na najnižem sloju nalaze se URI (engl. *Uniform Resource Identifier*) i IRI (engl. *Internationalized Resource Identifier*) koji se odnose na resurse samog Weba, te XML sa svojim prostorom imena (engl. *namespace*) i shemom koja pruža sintaktičke opise objekata[12]. XML se koristi za postizanje interoperabilnosti na Web-u. Pruža elementarnu

sintaksu za strukturiranje sadržaja unutar dokumenata, no ne daje nikakvo semantičko značenje samom sadržaju koji se prikazuje u dokumentu [15].

Drugi sloj je RDF i on se nalazi na istoj razini kao i XML, no obuhvaća i dio prostora iznad samog XML-a. RDF je W3C standard za opis resursa na Web-u, a pisan je u XML-u. Nije napravljen za ljudsku interpretaciju, već za interpretaciju od strane računala. Jednostavan je jezik za prikaz podatkovnih modela koji se odnose na resurse na Web-u i njihove međusobne odnose. RDFS (RDF Schema) proširuje RDF i to je jedna vrsta rječnika koja služi za opis svojstava i klasa iznad spomenutih resursa [15].

Treći sloj je ontološki sloj. Ontologija pruža kontrolirani rječnik koncepata od kojih svaki ima eksplicitno definirano značenje koje mogu procesirati računala. Definirajući zajedničku domenu, ontologije pomažu ljudima i računalima da međusobno komuniciraju [16]. Dakle, ontologije pružaju precizan opis objekata, njihovih svojstava i odnose s drugim objektima u nekoj domeni. Jedan od jezika koji je nastao za opsežnije opise Web resursa ili objekata je OWL (engl. *Web Ontology Language*) i to je jezik za procesiranje informacija na Web-u. Također spada u W3C standarde i dijeli se na tri podvrste: OWL Lite, OWL DL (*Description Logic*) i OWL Full. Osim što opsežnije opisuje same resurse, daje i detaljnije opise njihovih međuodnosa (kardinalnost, jednakost, različitost su samo neka od svojstava koja mogu opisivati odnose između klasa). U ovaj sloj pripadaju još i SPARQL (engl. *Simple Protocol and RDF Query Language*) i RIF (engl. *Rule Interchange Format*). SPARQL je protokol i jezik za pisanje upita nad podacima semantičkog Web-a. Upiti su bazirani na RDF podacima. RIF je W3C preporuka i pretežno se bazira na pravilima koja se koriste u ontologijama [15].

Četvrti sloj je logički sloj. Logika je potrebna za izražavanje pravila koja se koriste za dodatno zaključivanje. Ovaj sloj dodatno poboljšava jezike ontologije koji su spomenuti u prethodnom poglavlju [15].

Peti sloj, dokaz (engl. *Proof*) uključuje deduktivne procese i prikazuje dokaze Web jezika. Prema Berners – Lee-u, izvedba dokaza nije dio semantičkog Web-a no svejedno je vrlo aktivno područje kojeg se ne može standardizirati. Semantički Web koristi ovaj sloj samo za verifikaciju tih dokaza. Za primjer je prikazan sljedeći slučaj: ukoliko netko pošalje dokaz stranici X da je autoriziran za korištenje te stranice, tada stranica X mora biti u mogućnosti verificirati taj dokaz, a to je moguće uz pomoć alata za zaključivanje [15].

Šesti sloj je sloj povjerenja (engl. *Trust*). Povjerenje je važan dio semantičkog Web-a. Ukoliko tvrtka A pošalje neku informaciju tvrtki B, no tvrtka B ne može sa sigurnošću provjeriti da je ta informacija uistinu od tvrtke A, tada je ta informacija bezvrijedna. Povjerenje se postiže uporabom digitalnog potpisa [15]. Ukratko, digitalni potpis osigurava primatelju da

je poruka došla od baš od navedenog pošiljatelja te da nije ni u kojem trenutku bila izmijenjena.

Zadnji sloj čine korisničko sučelje i aplikacije. Ovaj sloj omogućuje interakciju korisnika sa sadržajem koji se nalazi na semantičkom Web-u. Osim čitanja sadržaja semantičkog Web-a, korisničko sučelje bi trebalo korisniku pružiti mogućnosti kreiranja i ažuriranja postojećih podataka [12].

Ovakva podjela na slojeve ima dvije funkcije, sprečavanje višeg sloja da implementira funkcionalnosti koje je već implementirao niži sloj, te omogućavanje aplikaciji koja razumije samo niži sloj da može interpretirati bar dio višeg sloja [12].

2.3.2. RDF

RDF ili *The Resource Description Framework* je okvir za prikaz informacija na Web-u [18]. Njegov cilj je omogućiti aplikacijama razmjenu podataka na Web-u bez promjene originalnog značenja tih podataka. Za razliku od HTML-a i XML-a, svrha RDF-a nije strukturirani prikaz podataka već daljnje procesiranje i obrada informacija koje ti podaci sadržavaju. RDF se smatra osnovnim formatom za razvoj semantičkog Web-a [11].

2.3.2.1. Uvod u RDF

RDF omogućuje objavu sadržaja na Webu o bilo čemu i može ga objaviti bilo tko. Pomoću RDF-a mogu se opisati Web resursi tog sadržaja kao što su autor, datum objave sadržaja, naslov, tema, autorska prava objavljene slike i ostalo [12].

Razvoj RDF-a započinje 90-ih godina. Prva verzija objavljena je 1999. godine od strane World Wide Web Consortium-a (W3C). Ova verzija RDF-a bila je primarno fokusirana na prikaz metapodataka Web stranica [11]. Nakon objave članka o semantičkom Web-u T. Berners – Lee-a, RDF je evolvirao do W3C standarda objavljenog 2004. godine čija svrha je još uvijek međusobno povezivanje podataka i pružanje interoperabilnosti [12].

Kao što je spomenuto, RDF doslovno znači okvir za opis resursa na Web-u. Svaki dio tog naziva može se i dodatno pojasniti:

- Resursi (engl. *resources*) su osnovni koncept na kojem se temelji semantički Web. To može biti sve što se nalazi na Web-u, npr. Web stranice, slike, videozapisi, osobe, mjesta, uređaji, događanja, organizacije, proizvodi ili usluge. Dakle, sve ono što se može identificirati URI-jem [12].
- Opisi resursa (engl. *description*) nužni su za njihovo razumijevanje i daljnje zaključivanje. Općenito, opis se može definirati kao skup atributa, značajki i odnosa određenog resursa [12].

- Okvir (engl. *framework*) pruža modele, jezike i sintaksu opisima resursa [12].

Ukratko, RDF pruža standardnu strukturu podataka i model za kodiranje podataka i metapodataka o bilo kojoj temi na Web-u. Ti anotirani podaci su tada povezani s drugim anotiranim podacima na Web-u i na taj način se stvara mreža povezanih resursa sa zajedničkim informacijama [12].

RDF se sastoji od nekoliko ključnih koncepata:

- Grafovi
- URI
- Tipovi podataka
- Literali
- Sintaksa za XML serijalizaciju
- Izražavanje jednostavnih izjava
- Dedukcija ili implikacija (engl. *entailment*)

2.3.2.2. RDF grafovi

Jedan od ključnih koncepata koje RDF koristi su grafovi [18]. RDF dokument opisan je pomoću usmjerenog grafa. Usmjereni graf se sastoji od skupa čvorova koji su povezani strelicama. Oba čvora i strelica koja ih povezuje imaju svoje identifikacijske oznake koje ih međusobno razlikuju. Slika 2. prikazuje primjer jednog takvog grafa [11].

Slika 2. RDF graf

Često se kaže da se RDF grafovi sastoje od RDF trojki (engl. *triples*). RDF trojke opisuju i povezuju objekte kombinacijom resursa, njihovih svojsatva i vrijednostima tih svojstava. Resurs je u tom slučaju subjekt neke izjave, svojstvo je predikat, a vrijednost tog svojstva je objekt izjave ili rečenice. Dakle, osnovna struktura podataka pisanih u RDF-u je *<subjekt, predikat, objekt>* [12]. Gledajući Sliku 2., može se reći da je Tim Berners - Lee subjekt, jeOsnivač je predikat, a Web je objekt.

2.3.2.3. URI

Jedan od problema koji se javlja je taj da nemaju svi RDF dokumenti na jednak način definirane identifikatore. Čak i kada postoje dva dokumenta koja obrađuju slične teme,

identifikatori koje koriste za neka dva pojma istog značenja mogu biti potpuno različiti. Drugi slučaj koji se javlja je korištenje istog identifikatora za pojmove koji imaju različita značenja [11]. Na primjer, ne postoji globalni identifikator za pojam Web-a. U jednom dokumentu on može biti nazvan „Web“, dok u drugom „WWW“, „World Wide Web“ ili nešto treće. Iako ti identifikatori imaju različite nazive, značenje im je isto. U drugom primjeru identifikator „HR“ može označavati državu Hrvatska ili ljudske resurse (engl. *Human Resources, HR*).

Da bi se riješio taj problem, RDF koristi URI. URI je zapravo generalizacija URL-a, svaki URL je također i URI. Za identifikaciju objekata u RDF-u se može koristiti i URL, no kako cilj nije razmjena podataka o različitim Web stranicama, već razmjena podataka o što više resursa koji se mogu pronaći na Web-u, oni se baš i ne koriste za njihovu identifikaciju. Ti resursi mogu biti bilo koji objekti koji imaju jasnu definiciju i način upotrebe (knjige, mjesta, ljudi, događaji, odnosi, apstraktni koncepti i ostalo). Nekima od tih resursa se ne može pristupiti putem URL-a pa je zato logično rješenje koristiti URI za njihovu identifikaciju [11].

Shema samog URI-ja prikazana je na Slici 3., a ispod slike su dana dodatna pojašnjenja o samim elementima od kojih se URI sastoji:

Slika 3. Shema URI-ja

- **Shema** (engl. *Scheme*) – naziv sheme koja specificira tip URI-ja, daje dodatne informacije o samom URI-ju i kako ga koristiti (npr. *http, ftp, mailto, file*…) [11]
- **Vlasništvo** (engl. *Authority*) – vlasništvo na Web-u se obično odnosi na ime domene ili korisnika s dodatnim detaljima o portu. (npr. *W3.org, john@example.com, example.org:8080*…). Ovaj dio je opcionalan i prepoznaje se po prefiksu „//“. [11]
- **Putanja** (engl. *Path*) – putanja je glavni dio većine URI-ja, iako je moguće koristiti i prazne putanje kao na primjer u email adresi. Putanje se mogu hijerarhijski sortirati koristeći znak „/“ kao separator (npr. */examples/example1*) [11]
- **Upit** (engl. *Query*) – upit je također neobavezni dio URI-ja koji može pružiti dodatne informacije o resursu. Prepoznaje se po sufiksu „?“ i obično se koristi za definiranje dodatnih parametara web servisima (npr.*q=Semantic+Web*) [11]
- **Fragment** (engl. *Fragment*) – još jedan neobavezni dio koji pruža način za identifikaciju resursa. Prepoznaje se po prefiksu „#“. URI-ji s različitim fragmentima upućuju na različite resurse iako se oni možda nalaze na istoj web stranici (npr. *section1*) [11]

Za URI se koriste osnovni znakovi latiničnog pisma uključujući i brojeve, no ukoliko je potrebno koristiti neke druge znakove, kao što su na primjer dijakritički znakovi, to je također dozvoljeno i u tom slučaju se URI naziva IRI i može se koristiti na svim područjima kao i URI [11].

2.3.2.4. Tipovi podataka

RDF koristi tipove podataka kako bi razlikovale vrijednosti poput brojeva, stringova i datuma. Tipovi podataka sastoje se od leksičkog prostora (engl. *Lexical space*), vrijednosnog prostora (engl. *Value space*) i mapiranja iz leksičkog u vrijednosni prostor (engl. *Lexical-to-value mapping*). Leksički prostor tipa podatka je skup Unicode stringova, vrijednosni prostor prikazuje vrijednost podatka, a mapiranje tipova podataka je zapravo skup parova čiji prvi element pripada leksičkom prostoru, a drugi element pripada vrijednosnom prostoru tipa podatka. Svaki element leksičkog prostora ima svoj određeni par koji ima točno jednu vrijednost [18].

Primjer gore opisanih prostora može se dodatno objasniti na tipu podatka XML Scheme *xsd:boolean* [20]:

- Leksički prostor: *„true“, „false“, „1“, „0“*
- Vrijednosni prostor: ***true, false***
- Mapiranje iz leksičkog u vrijednosni prostor: <*“true“,* ***true>***, <*“false“,* ***false***>, <*“1“,* ***true***>, <*“0“,* ***0***>

Tipovi podataka se kao i ostali resursi identificiraju pomoću URI-ja. Većina tipova podataka definirana je u XML Schemi [11]. XML Schema ili XML Schema Definition (XSD) se koristi za opis i validaciju strukture i sadržaja XML podataka. Pomoću nje se definiraju elementi, atributi i tipovi podataka [21]. Neki od XSD tipova podataka koji su kompatibilni s RDF-om su osnovni tipovi poput *xsd:string, xsd:boolean, xsd:decimal, xsd:integer* i drugi. Osim tipova podataka koji se nalaze u XML Schemi, postoje i tipovi podataka specifični za HTML, *rdf:HTML* i tipovi podataka koji omogućuju prikaz XML sadržaja, *rdf:XMLLiteral* [20].

2.3.2.5. Literali

Literali se koriste za identifikaciju vrijednosti poput brojeva i datuma. Sve što se prikazuje literalima, može se prikazati i URI-jem, no ovisno o slučaju, literali mogu biti bolji izbor. Literali su objekt RDF izjave i oni nikad ne mogu biti predikat ili subjekt [18].

Postoje dvije vrste literala: običan ili jednostavan literal (engl. *plain literal*) i tipizirani literal (engl. *typed literal*). Obični literal je string koji može, ali i ne mora uključivati oznaku s jezikom u kojem je pisan i koristi se za obični tekst u prirodnom govoru. Tipizirani literal je

također string, no on sadrži i tip podatka pisan pomoću URI-ja. Primjer jednog tipiziranog literala je <*xsd:boolean, „true“*> [18].

2.3.2.6. Sintaksa za XML serijalizaciju

Jedan od načina za prikaz podataka u RDF-u su grafovi, no postoji li veći broj podataka s više od tisuću različitih čvorova, takav slikovni prikaz podataka nije idealan za računala. Zbog toga se uvodi novi način reprezentacije RDF-a pomoću niza znakova koji se lako mogu zapisati u bilo koji dokument na računalu. Da bi to bilo moguće, potrebno je graf razdvojiti na manje dijelove koji se mogu spremati u taj dokument dio po dio. Taj proces transformacije složenih struktura podataka u linearne nizove podataka naziva se serijalizacija [11].

Ti manji dijelovi na koje je RDF graf razdijeljen nazivaju se trojke i o njima je već bilo riječi u prethodnim poglavljima. Ukratko, trojke se odnose na subjekt, predikat i objekt RDF izjave.

Jedna od najčešćih i najpoznatijih sintaksi za XML serijalizaciju RDF-a je RDF/XML serijalizacija. XML pruža sintaktičku strukturu za organizaciju RDF dokumenta i ona je hijerarhijska, pa stoga i trojke RDF-a grafa sad moraju biti hijerarhijski organizirane. Primjer RDF/XML serijalizacije na temelju grafa sa Slike 2.:

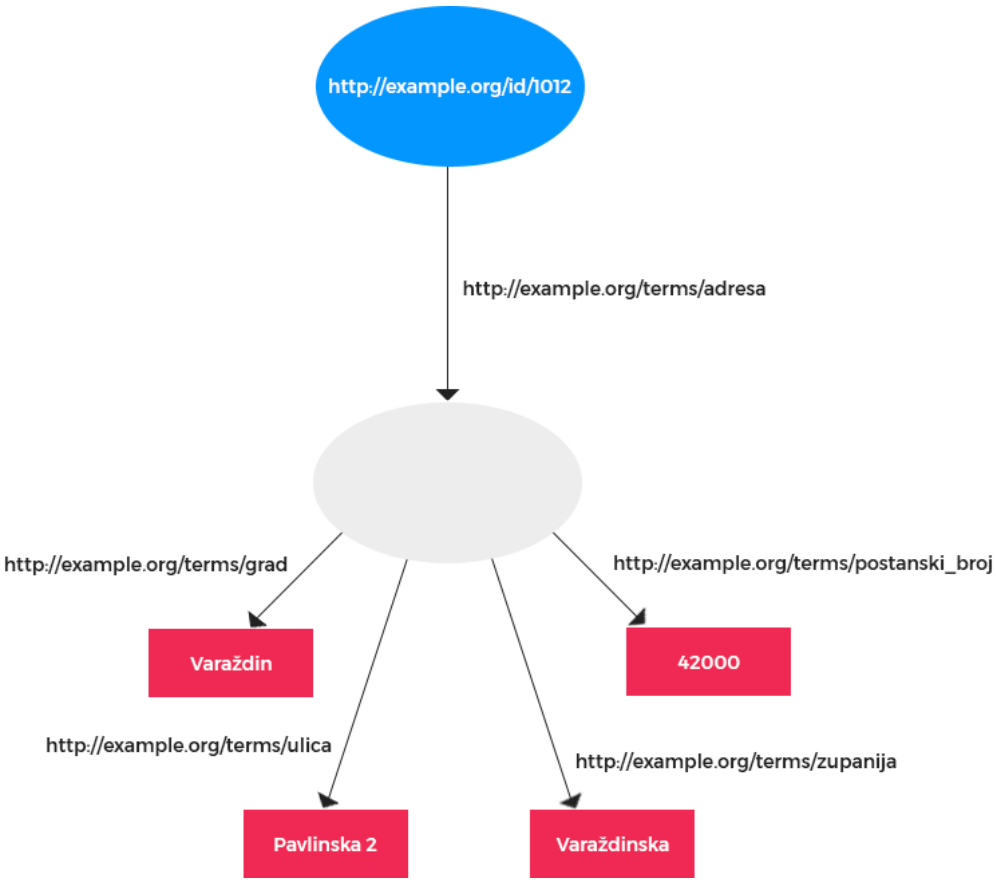
```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ex ="http://example.org/">

<rdf:Description rdf:about="http:// www.w3.org/People/Berners-Lee/">
<ex:jeOsnivac>
<rdf:Description rdf:about="http://web/uri">
</rdf:Description>
</ex:jeOsnivac>
</rdf:Description>
</rdf:RDF>
```

2.3.2.7. Izražavanje jednostavnih izjava

Jednostavne izjave izražavaju vezu između dva objekta. Izjave, kao što je rečeno, mogu biti izražene i kao RDF trojke gdje je predikat naziv te veze, a subjekt i objekt su zapravo objekti koji se povezuju. Još jedan način za prikaz jednostavnih izjava je i tablicom, gdje se gleda redak s dva stupca i jedan stupac predstavlja subjekt, a drugi objekt. Naziv tablice tada odgovara predikatu, tj. nazivu same veze [20].

Relacijske baze podataka dopuštaju da tablica ima proizvoljan broj stupaca i redaka, no oni trebaju biti dekomponirani da bi mogli odgovarati RDF trojkama. Za taj proces dekompozicije uvodi se prazan čvor (engl. *blank node*) koji predstavlja jedan redak, a trojka je zapravo svaka ćelija u tom retku. Subjekt je tad taj prazan čvor, predikat je isti kao i naziv stupca, a objekt je vrijednost ćelije. Primjer ove reprezentacije prikazan je na Slici 4. [20].

Prazni čvorovi ne mogu biti globalno prikazani URI-jem i ne sadržavaju nikakve dodatne informacije. Također, praznim čvorovima se mogu prikazati samo subjekti i objekti RDF trojki, jer predikati moraju obavezno biti imenovani URI-jem [11].



Slika 4. Korištenje praznog čvora (prema [20])

Ukratko, RDF je evoluirao do W3C preporučenog standarda za opis, objavu i povezivanje metapodataka na Web-u. On identificira URI-je korištenih resursa i omogućuje prikaz veze između dva ili više resursa. Ponovnim korištenjem URI-ja dana je mogućnost korisnicama da objavljuju podatke o raznim temama i svatko može nadodati što želi na tu temu. Razvoj RDF-a pridonio je i razvoju RDF Scheme koja sadržava opise klasa RDF resursa, njihova svojstava i opise veza kojima su ti resursi povezani [11]. Više riječi o RDF Schemi u sljedećem poglavlju.

2.3.3. RDFS

RDF Schema ili RDFS je dio W3C RDF standarda i nije ništa drugo no još jedan RDF rječnik. Također, svaki RDFS dokument je također i RDF dokument, što osigurava da je RDFS kompatibilan sa svim alatima koji podržavaju i RDF [11].

RDFS ima svoj imenski prostor (engl. *namespace*) i on glasi *http://www.w3.org/2000/01/rdf-schema#* no često je skraćen samo na *rdfs:*. RDFS se još naziva i ontološkim jezikom. Već prije je spomenuto da je ontologija opis neke domene koja zanima korisnika te da je to znanje obradivo od strane računala i ima formalno definirano značenje. Tako je i RDFS obradiv od strane računala i sadržava opis neke domene [11].

RDFS se još naziva i *lightweight* jezikom za deklaraciju i opis tipova resursa (klase), te za opis veza između resursa i tipova atributa (svojstva). Omogućuje i definiranje naziva klasa postojećih resursa, tipova veza između instanci tih klasa i tipova resursa koje te veze povezuju. Semantika RDFS-a se temelji na skupovima i njihovim operatorima (unija, presjek i uključivanje). Setovi su zapravo resursi koji pripadaju nekoj klasi ili resursi koji pripadaju nekoj relaciji. RDFS je rječnik za opis veza između sljedećih setova: veze između klasa, veze između svojstava i veze između klasi i svojstava [12].

Koncepti RDFS-a su klase i svojstva. Svaki je ukratko opisan u sljedećim odlomcima.

2.3.3.1. Klase

Resursi su podijeljeni u grupe koje se nazivaju klase. Svi članovi te grupe ili klase nazivaju se instancama klase. Može se reći da su klase zapravo resursi. Klase se često identificiraju pomoću IRI-ja, a opisuju se RDF svojstvima. Da bi prikazali da je neki resurs instanca klase, koristi se oznaka *rdf:type*, dok se skup resursa koji su ujedno i RDFS klase označava pomoću *rdfs:Class* [22].

Osim *rdfs:Class* postoje i drugi nazivi klasa kao što su [11]:

- *rdfs:Resource* – označava klasu svih resursa (npr. svi elementi koji pripadaju nekoj domeni)
- *rdfs*:*Property* – odnosi se na klasu svih svojstava resursa
- *rdfs*:*XMLLIteral* – označava klasu svih vrijednosti tipa *XMLLiteral*
- *rdfs:Literal* – označava klasu svih literala, što također podrazumijeva da obuhvaća tipove podataka kao potklase
- *rdfs:Datatype* – elementi ove klase su zapravo tipovi podataka, kao što je npr. XMLLiteral
- *rdf:Bag, rdf:Alt, rdf:Seq, rdfs:Container* – služe za deklaraciju spremnika

- *rdfs:ContainerMembershipProperty* – označava klasu koja sadrži instance svojstva nekog resursa i njima se izražava da je neki resurs član spremnika
- *rdf:List* – označava klasu svih kolekcija
- *rdf:Statement* – odnosi se na klasu reificiranih trojki

Važno je napomenuti da resurs može pripadati više različitih klasa, a ne samo jednoj [11]. Također, *rdfs:Class* se može samo-referencirati, što znači da klasa *rdfs:Class* može pripadati samoj sebi. Iako je standardno da je klasa skup elemenata koji joj pripadaju, u RDF Schemi je kolekcija svih klasa zapravo sama klasa [22].

Uz klase postoje i potklase. Potklase imaju oznaku rdfs:subClassOf i označavaju da je neka klasa potklasa druge. Upotrebom potklasa moguće je definirati hijerarhiju klasa i specificirati njihovu generalizaciju/specifikaciju u nekoj domeni. Dokumenti koji sadrže isključivo hijerarhiju klasa nazivaju se taksonomije, dok se veze potklasa – superklasa obično nazivaju taksonomskim vezama [11].

2.3.3.2. Svojstva

URI-ji predikata imaju posebnu ulogu u RDF trojkama. Iako URI-ji obično predstavljaju resurse, oni mogu biti i u ulozi predikata (npr. *ex:jeNapisao, ex:ima, ex:seSastoji*, itd.). Kada su URI-ji u ulozi predikata neispravno ih je smatrati resursima, već se na njih gleda kao na svojstva koja opisuju vezu između dva resursa. Da bi se označilo da je neki URI svojstvo, koristi se oznaka *rdf:Property.* To je zapravo klasa svih svojstava [11].

Kao i kod klasa i potklasa, RDFS ima mogućnost kreiranja podsvojstava. Podsvojstva se označavaju pomoću *rdfs:subPropertyOf*. Slijedi primjer jednog podsvojstva: *ex:jeSretnoOzenjen* je podsvojstvo svojstva *ex:jeOzenjen*. Dakle, ako vrijedi neko podsvojstvo, tada vrijedi i njegovo svojstvo, isto kao i kod klasa i potklasa [11].

Također, RDF ne zahtjeva da su klase i svojstva odvojeni (engl. *disjoint*) jedno od drugog. Neki resurs može biti i klasa i svojstvo, iako je u tom slučaju potrebno koristiti različite resurse za klasu i svojstvo [22].

RDFS može navesti da svi subjekti i objekti nekog svojstva pripadaju određenoj klasi. Za određivanje svih subjekata koji pripadaju nekoj klasi koristi se oznaka *rdfs:domain.* Na primjer, ako postoji svojstvo *ex:jeMajkaOd* i ono ima domenu *ex:Zena* i *ex:Roditelj*, tada se podrazumijeva da svi subjekti koji imaju to svojstvo pripadaju klasama *ex:Zena* i *ex:Roditelj*. Slično vrijedi i za doseg (engl. *range*) svojstvo. Ono navodi kako svi objekti nekog svojstva pripadaju određenoj klasi ili klasama [22].

Domena i doseg su ograničenja svojstva i ona predstavljaju tzv. semantičku vezu između klasa i svojstava jer je to jedini način za opis željene međuovisnosti između različitih elemenata [11].

2.3.3.3. Ostale strukture podataka

RDFS podržava i neke strukture podataka kao što su liste. Liste se u RDFS-u prikazuju na dva načina: kao spremnici (engl. *containers*) i kao kolekcije (engl *collections*).

Spremnici su definirani oznakom *rdfs:Container* i oni sadržavaju resurse ili literale. Postoje tri potklase spremnika koje su spomenuta prilikom nabrajanja različitih klasa u RDFS-u, a to su *rdfs:Seq, rdfs:Bag* i *rdfs:Alt.* Sve tri potklase su poredane liste, no *rdfs:Seq* se obično koristi u situacijama gdje je poredak bitan, dok se *rdfs:Bag* koristi u slučaju kad poredak elemenata nije bitan. Treća vrsta spremnika *rdfs:Alt* predstavlja listu alternativa. Elementi u spremniku su obično definirani na sljedeći način: *rdf:li* gdje je *li* zapravo indeks elementa (npr. *rdf:_1* je prvi element, *rdf:_2* je drugi elementi, itd.) [22].

Drugi način za prikaz liste u RDFS-u je kolekcija. Kolekcije su napravljene s namjerom da se prevladaju dva nedostatka spremnika. Prvi nedostatak je taj da ne postoji način da se spremnik zatvori, a spremnik bi se trebao zatvoriti nakon što nema više novih elemenata. Drugi nedostatak je taj da su programeri navikli koristiti liste za prikaz kolekcija [22].

2.3.3.4. Reifikacija

Kod RDFS-a je bitno spomenuti i reifikaciju: izjavu o izjavama. Kao što je već navedeno, RDF(S) je osmišljen za predstavljanje informacija na Web-u. Te informacije mogu biti točne ili netočne. Da bi neka aplikacija utvrdila hoće li vjerovati informacijama na Web-u, ona može provjeriti podrijetlo tih informacija i otkriti njihov izvor. Taj proces se naziva reifikacijom [22].

Reificirana izjava je zapravo resurs koji predstavlja pojavu RDF izjave. Takva izjava pripada klasi *rdf:Statement* i sastoji se od subjekta *rdf:subject*, predikata *rdf:predicate* i objekta *rdf:object* izjave. Reificiranoj izjavi mogu biti pridodana i ostala svojstva kao što su mjesto gdje se izjava pojavljuje, tko je odgovoran za tu izjavu i ostalo [22].

Iako reifikacija postoji, njena upotreba i nije baš česta, te se čak i obeshrabruje. Takve izjave su često nejasne i prilično velike, te upotreba reificiranih izjava nije isto kao što i upotreba originalne izjave i nijedna izjava ne podrazumijeva drugu. Planira se uvesti koncept grafova i imenovanih grafova u RDF-u koji bi imali mogućnost dodavanja izjava o samim grafovima, te bi time i nestala potreba za reifikacijom [12].

2.3.4. OWL

OWL ili Web Ontology Language je W3C standard za modeliranje ontologija [11]. No prije upoznavanja s njegovom sintaksom i mogućnostima, potrebno je objasniti sam pojam ontologije.

2.3.4.1. Definicija ontologije

Pojam ontologije ima različita značenja ovisno o kontekstu u kojem se upotrebljava. Sam pojam potječe iz polja filozofije i metafizike te se odnosio na studije o samom postojanju bića i stvari. Ontologije pružaju detaljne klasifikacije o objektima i njihovim vezama u svim sferama postojanja [24].

Definicija ontologije u računalnim znanostima se nešto razlikuje od prijašnje. Ontologija je u ovom slučaju kolekcija definicija o pojedinoj domeni koja se sastoji od različitih objekata [12]. Ona opisuje prikupljeno znanje o određenoj domeni i ono je predstavljeno na način da može biti procesirano od strane računala i ima formalno određeno značenje [11]. Najšire prihvaćena definicija ontologije je ona koju je dao Thomas Gruber: „Ontologija je formalna specifikacija konceptualizacije“. Pojam specifikacije odnosi se na objekte, koncepte i ostale entitete za koje se pretpostavlja da postoje unutar neke domene, te na njihove međusobne veze, dok se pojam konceptualizacije odnosi na apstraktni model te domene koji identificira same koncepte (klase) od kojih se domena sastoji [24, 26].

Također, dana je i definicija ontologije u kontekstu OWL-a. U ovom slučaju, ontologija pruža način za definiranje klasa, svojstva, individua i njihovih međusobnih veza [24].

2.3.4.2. Definicija OWL-a

Iako RDFS ima klase i svojstva i pomoću njega se mogu modelirati neke jednostavne ontologije, za modeliranje kompleksnijeg znanja na temelju formalne logike uz mogućnost zaključivanja nastaje novi jezik, OWL [11, 25].

OWL je jezik koji je nastao u svrhu definiranja ontologija i reprezentacije informacija koje su razumljive ljudima ali ih mogu obraditi i računala. Koristi se za davanje izjava o klasama, svojstvima i individuama. Te tvrdnje ili izjave mogu biti sadržane u jednoj ontologiji ili mogu biti prikazane kombinacijom više različitih, ali spojenih ontologija [24]. Temelji se na opisnoj logici (engl. *Description Logic*) [12].

OWL je u potpunosti integriran u semantički Web i koristi ostale W3C standarde. Službena sintaksa mu se temelji na RDF/XML sintaksi. Nazivi resursa kao što su klase, individue i ostalo definiraju se pomoću IRI-ja. Za definiranje tipova podataka i vrijednosti koristi XML Schema tipove podataka. Same OWL ontologije se smatraju Web dokumentima, te se spremaju i koriste kao i ostali Web dokumenti. Također, OWL ima mogućnost korištenja

konstrukata za identificiranje različitih ontologija, kao i za uvoz jedne ontologije u drugu u svrhu spajanja dvije ili više ontologija. Ontologije se mogu kombinirati s drugim ontologijama koristeći i novi W3C Rule Interchange Format (RIF) standard [12].

2.3.4.3. Vrste OWL-a

OWL se može podijeliti u tri vrste ili podjezika: OWL Full, OWL DL i OWL Lite.

OWL Full sadrži preostala dva podjezika, OWL DL i OWL Lite. Jedini je od tri vrste koji sadržava sve dijelove RDFS-a i u potpunosti je uzlazno kompatibilan s RDF-om. Vrlo je izražajan. Semantički je nešto teže razumljiv od ostalih podjezika i zbog svoje izražajnosti je neodlučiv, što utječe na mogućnost zaključivanja. Svaki valjani RDF dokument je također i valjani OWL Full dokument i svaki valjani RDF/RDFS zaključak je valjani OWL Full zaključak [11, 25].

OWL DL (Description Logic) sadržava OWL Lite. Za razliku od prijašnjeg podjezika, OWL DL je odlučiv pa i s time provodi bolje zaključke. Podržava ga većina softverskih alata [11]. Jedan od nedostataka je taj da nije u potpunosti kompatibilan s RDF-om, što znači da ako neki RDF dokument želi biti valjani OWL DL dokument, potrebno ga je proširiti u nekim dijelovima i ograničiti u drugim, no svaki OWL DL dokument je valjani RDF dokument [25].

OWL Lite je sadržan u prijašnja dva podjezika. Odlučiv je i manje izražajan [11]. Korisnici ga lakše savladavaju i lakši je za implementaciju. Jedan od nedostataka je njegova restriktivna izražajnost. Ne podržava prebrojive klase, izjave o razdvojenim klasama i dopušta samo neka ograničenja kardinalnosti [25] .

Koji od ovih podjezika koristiti ovisi isključivo o programeru i njegovim potrebama. Postoje stroga pravila o uzlaznoj kompatibilnosti između tri podjezika [25]:

- Svaka valjana OWL Lite ontologija je i valjana OWL DL ontologija
- Svaka valjana OWL DL ontologija je i valjana OWL Full ontologija
- Svaki valjani OWL Lite zaključak je i valjani OWL DL zaključak
- Svaki valjani OWL DL zaključak je i valjani OWL Full zaključak

2.3.4.4. OWL dokumenti i zaglavlje

OWL se temelji na RDF I RDF Schemi i koristi RDF/XML sintaksu koja i nije baš čitljiva običnim korisnicima. Zbog toga postoji i nekoliko drugih sintaksi kao što su sintaksa koja se bazira na XML-u i ne prati RDF konvencije, te je zbog toga puno lakša za čitati od RDF/XML sintakse. Druga korištena sintaksa je grafička sintaksa koja se bazira na UML (*Universal Modelling Language*) jeziku koji je često u upotrebi i time olakšava korisnicima upoznavanje s OWL-om [25].

Kao što je spomenuto, OWL dokumenti se još nazivaju i OWL ontologije i to su zapravo RDF dokumenti. U zaglavlju svake ontologije nalazi se korijenski *rdf:RDF* element koji određuje broj imenskih prostora. Primjer jednog takvog zaglavlja [25]:

```
<rdf:RDF
xmlns:owl ="http://www.w3.org/2002/07/owl#"
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd ="http://www.w3.org/2001/XLMSchema#">
```

OWL dokument može sadržavati i neke dodatne informacije o ontologiji koje su sadržane unutar *owl:Ontology* elementa. Moguće je i uvesti različite ontologije koristeći *owl:imports* element i tada se uvezena ontologija smatra dijelom originalne ontologije [11]. Element za uvoz drugih ontologija je tranzitivan, što znači da ako ontologija A uveze ontologiju B i ontologija B uveze ontologiju C, tada ontologija A ujedno uvozi i ontologiju C [25].

2.3.4.5. Klase

Klase se u OWL-u definiraju pomoću oznake *owl:Class.* OWL klase su potklase RDF klasa. Klase mogu biti međusobno razdvojene (engl. *disjoint*) ili ekvivalentne (engl. *equivalent*) jedna drugoj. Npr. klasa *Profesor* je razdvojena od klase *Asistent,* dok su klase *Web* i *World Wide Web* ekvivalentne. Razdvojene klase označene su *owl:disjointWith* elementom, a ekvivalentne s *owl:equivalentClass* elementom. Postoje i dvije predefinirane klase, *owl:Thing* i *owl:Nothing*. Prva klasa je općenita klasa koja sadrži sve ostale klase u ontologiji, a druga klasa je prazna klasa [25].

2.3.4.6. Svojstva

Postoje tri vrste svojstva u OWL-u: objektna svojstva (engl. *object properties*) koja povezuju objekte s drugim objektima (npr. *predaje*), podatkovna svojstva (engl. *datatype properties*) koja povezuju objekte s podatkovnim vrijednostima (npr. *dob, telefon*…) i anotacijska svojstva (engl. *annotation properties*) koja ontologiji, aksiomima ili IRI-ju daju bilješke ili anotacije koje mogu biti različitog tipa: oznaka (engl. label), komentar (engl. comment), drugi IRI i ostalo [25, 45].

OWL ima mogućnost i definiranja inverznih svojstava. Ukoliko postoji svojstvo *predaje*, njegovo inverzno svojstvo je *jePredavanOd*. Inverzna svojstva označavaju se elementom *owl:inverseOf* [25].

Svojstva mogu imati i svoja ograničenja. Element *owl:allValuesFrom* specificira sve klase koje mogu imati svojstvo definirano elementom *owl:onProperty*. Drugim riječima, sve vrijednosti nekog svojstva moraju pripadati točno određenoj klasi. Još jedno ograničenje je *owl:hasValue.* Ono definira specifičnu vrijednost koju svojstvo mora imati [25].

Spomenuvši ograničenje *owl:allValuesFrom*, postoji i slično ograničenje *owl:someValuesFrom*. Razlika je u tom što je prvo ograničenje zapravo univerzalni kvatifikator, a drugo je egzistencijalni kvantifikator. Primjer upotrebe ograničenja *allValuesFrom* je sljedeći: postoji klasa *Kolegij* i svojstvo *jePredavanOd*. To svojstvo ima spomenuto ograničenje za klasu *Profesor.* Dakle, **sve vrijednosti** svojstva *jePredavanOd* moraju pripadati klasi *Profesor.* Sličan primjer vrijedi i za ograničenje *someValuesFrom*. Postoji klasa *Profesor* i ona ima spomenuto ograničenje nad svojstvom *predaje* za klasu *PreddiplomskiPredmet.* To znači da **barem jedna** vrijednost svojstva *predaje* mora pripadati klasi *PreddiplomskiPredmet* [25].

Osim ograničenja opisanih u odlomku iznad, postoje i ograničenja kardinalnosti. Ona definiraju točan broj ili rang kojem vrijednosti nekog svojstva moraju pripadati. Ograničenja kardinalnosti također se definiraju unutar *owl:Restriction* elementa i mogu imati minimalnu kardinalnost (*owl:minCardinality*) i/ili maksimalnu kardinalnost (*owl:maxCardinality*). Kardinalnosti se definiraju prirodnim brojevima i nulom te je potrebno definirati tip podatka kao *&xsd;nonNegativeInteger*. Točno određena kardinalnost može se specificirati upotrebom istog broja za minimalnu i maksimalnu kardinalnost ili upotrebom elementa *owl:cardinality* [25].

Postoje i neka posebna svojstva kao što su [25]:

- **Tranzitivno svojstvo** – tranzitivne vrijednosti, „je majka od“, „je predak od“, „ima bolju ocjenu od“…
- **Simetrično svojstvo** – simetrične vrijednosti, „ima istu ocjenu kao“, „je prijatelj“…
- **Funkcionalno svojstvo** – ima jednu jedinstvenu vrijednost za svaki objekt, „dob“, „visina“…
- **Inverzno funkcionalno svojstvo** – jedinstveno svojstvo, postoji samo jedan subjekt koji je povezan s nekim objektom preko predikata koji nosi ovo svojstvo, npr. „OIB“, „JMBAG“… [44]

2.3.4.7. Boolean operatori

Za izražavanje kompleksnijeg znanja OWL pruža jezične elemente za konjunkciju, disjunkciju i negaciju, tj. *owl:intersectionOf, owl:unionOf, owl:complementOf* respektivno.

Konjunkcija dviju klasa sastoji se od točno tih objekata koji pripadaju i jednoj i drugoj klasi, disjunkcija dviju klasa podrazumijeva sve objekte koji pripadaju tim klasama, dok negacija obuhvaća samo one objekte koji se ne nalaze u određenoj klasi [11, 25].

2.3.4.8. Enumeracije

Enumeracije ili prebrojive klase se koriste za definiranje klase nabrajanjem svih njezinih elemenata. Oznaka za prebrojivu klasu je *owl:oneOf* [25].

2.3.4.9. Instance

Kao i u RDF-u, OWL ima instance klasa. OWL nije usvojio pretpostavku o jedinstvenim nazivima instanci, što znači da ako postoje dvije instance s različitim imenima, to ne znači da su i one dvije različite individue. Da bi se eksplicitno naglasilo da su dvije individue različite, potrebno je koristiti oznaku *owl:differentFrom*. Ukoliko postoji velik broj individua i potrebno je naglasiti da se one međusobno razlikuju, postoji i mogućnost korištenja oznake *owl:distinctMembers* unutar koje se tada nabrajaju sve međusobno različite individue. Potrebno je još i naglasiti da se ova oznaka nalazi unutar oznake *owl:AllDifferent* i da ona pripada jednoj od struktura podataka, kolekciji [25].

2.3.5. SPARQL

U prijašnjim poglavljima spomenuto je nekoliko načina za prikaz strojno – čitljivih informacija na Web-u. Sada, kad su te informacije prikazane, treba im na neki način i pristupiti. Za pristup tim informacijama koriste se upiti. Jedan od jezika za kreiranje upita koji se baziraju na RDF zapisu informacija zove se SPARQL ili *Simple Protocol And RDF Query Language* [11].

Jezik je nastao od strane W3C-a za kreiranje upita nad RDF podacima koji su objavljeni na Web-u, a sintaktički je sličan SQL jezicima [12]. Jezgra SPARQL-a su jednostavni upiti u obliku jednostavnih grafičkih uzoraka. SPARQL ima mogućnost kreiranja funkcija za složenije upite, te dodavanje uvjeta za filtriranje i formatiranje konačnih rezultata upita [11].

SPARQL je u potpunosti integriran u semantički Web i koristi ostale W3C standarde. Pretpostavlja da su podaci nad kojima se vrše upiti prikazani u obliku RDF grafova, da su resursi identificirani IRI-jima, a literali određeni XML Schema tipovima podataka. Iako je spomenuto da SPARQL sliči SQL jezicima, on je puno jači od samog SQL-a jer je kreiran za Web koji je po definiciji otvoreniji, decentralizirani i fluidniji od samih baza podataka [12].

2.3.5.1. Osnovni obrasci SPARQL upita

SPARQL ima četiri osnovna obrasca za kreiranje upita: SELECT, CONSTRUCT, DESCRIBE i ASK [12]:

- **SELECT** vraća sve varijable ili podskup varijabli definiranih u upitu [28]
- **CONSTRUCT** vraća rezultate u obliku RDF grafa [28]
- **DESCRIBE** vraća samo jedan rezultat u obliku RDF grafa koji sadržava i RDF podatke o resursima [28]
- **ASK** vraća vrijednosti *yes* u slučaju kad odgovor na upit postoji ili *no* kad odgovor na upit ne postoji, ne vraća nikakve dodatne informacije [28]

SELECT i CONSTRUCT obrasci su pogodni za upite nad poznatim krajnjim točkama (engl. *endpoints*). SPARQL, osim što je jezik za upite, uključuje i jednostavan komunikacijski protokol koji klijent može koristiti za upite nad udaljenim krajnjim točkama i dobivanje rezultata u XML-u ili RDF-u. SELECT obrazac vraća rezultate u obliku tablice u XML formatu, a CONSTRUCT obrazac vraća rezultate u RDF-u. CONSTRUCT obrazac se može koristiti i za pretvorbu jednog rječnika u drugi i na taj način se rješava problem heterogenih rječnika koji su dio Web-a.Također, ukoliko klijent ne zna rječnik nekog resursa, može koristiti i njegov IRI, te se u SPARQL upitu koristi DESCRIBE obrazac koji, kao što je spomenuto, vraća RDF graf koji opisuje traženi resurs [12].

Primjer jednog jednostavnog upita:

```
PREFIX ex:  <http://example.org/>
SELECT ?naziv ?autor
WHERE { ?knjiga   ex:jeIzdana <http://izdavac/algoritam/uri> .
           ?knjiga      ex:naziv    ?naziv .
           ?knjiga      ex:autor    ?autor }
```

Upit se sastoji od tri glavna djela označena s ključnim riječima koje se pišu velikim slovima, PREFIX, SELECT i WHERE. Svaki dio je posebno objašnjen u sljedećim odlomcima.

PREFIX označava prefiks imenskog prostora i deklaraciju nije potrebno završiti točkom [11].

SELECT obrazac određuje opći format prikaza rezultata. Izjave koje dolaze nakon SELECT obrasca odnose se na ostatak upita. Nazivi varijabli imaju prefiks „?“. U primjeru iznad, varijable su ?naziv i ?autor te one vraćaju nazive knjiga i autora [11].

WHERE započinje sam upit i nakon njega slijede jednostavni grafički obrasci, tj. trojke (kao kod RDF grafova). Ti obrasci su sadržani unutar vitičastih zagrada. U ovom slučaju,

WHERE obrazac se sastoji od tri trojke u Turtle notaciji (Turtle notacija može sadržavati i varijable, a ne samo URI-je). Varijable predstavljaju vrijednosti koje se koriste prilikom odgovaranja na upit i mogu se koristiti na više mjesta u upitu. Svaka trojka u WHERE obrascu mora na kraju imati točku. Točka se koristi kao separator trojki [11].

Dakle, upit u primjeru iznad dohvaća sve knjige čiji izdavač je Algoritam i imaju poznati naslov i autora. Rezultat se prikazuje u obliku parova naziv knjige i autor, jer su samo te dvije varijable definirane u SELECT obrascu. Da je umjesto naziva varijabli stavljen asterisk (*), upit bi dohvaćao sve poznate podatke o tim knjigama.

2.3.5.2. Modifikatori upita i uzorci

Upiti vraćaju rezultate u obliku neuređene liste. Da bi se kontrolirao format, poredak ili broj rezultata koriste se modifikatori upita [28]. Modifikatori upita izmjenjuju rješenje u WHERE djelu upita na način da početno, neuređeno rješenje upita uzimaju kao ulazne podatke, manipuliraju tim podacima ovisno o vrsti modifikatora koji se koristi i generiraju novo rješenje kao izlaz [12]. Postoji nekoliko vrsta modifkatora:

- **Modifikator poretka** (engl. *order modifier*) – generira uređenu listu rezultata upita. Ključna riječ modifikatora je ORDER BY i može se definirati rastući ili padajući poredak rezultata ključnim riječima ASC() ili DESC() respektivno. Ukoliko se ne specificira vrsta poretka, za zadanu vrstu koristi se ASC() ili rastući poredak [12, 28].
- **Modifikator projekcije** (engl. *projection modifier*) – prikazuje samo određene varijable u rezultatu nekog upita. Modifikator se definira nakon SELECT obrasca [12, 28].
- **Jedinstven modifikator** (engl. *distinct modifier*) – eliminira sve duplikate iz konačnog rezultata upita. Kao i prijašnji modifikator, dolazi nakon SELECT obrasca, a ključna riječ je DISTINCT [12, 28].
- **Modifikator redukcije** (engl. *reduced modifier*) – dopušta eliminaciju nekih duplikata iz konačnog rezultata upita. Ključna riječ modifikatora je REDUCED i dolazi nakon SELECT obrasca, ako i DISTINCT [12, 28].
- **Offset modifikator** (engl. *offset modifier*) – određuje broj rezultata koji će se preskočiti i neće se prikazati u konačnom rezultatu upita. Ključna riječ modifikatora je OFFSET i nakon nje se definira cijeli broj rezultata koje korisnik želi preskočiti [12, 28].
- **Modifikator limita** (engl. *limit modifier*) – određuje maksimalan broj rezultata koji se prikazuju u konačnom rješenju upita, a ključna riječ modifikatora je LIMIT nakon koje dolazi broj rezultata koje korisnik želi prikazati [12, 28].

Osim modifikatora upita postoje i neki uzorci (engl. *patterns*) koji se upotrebljavaju u jeziku SPARQL. Neki uzorci poput RDF terma, trojki i osnovnih grafičkih uzoraka već su spomenuti u ovom radu, no postoji još nekoliko uzoraka koji se mogu upotrebljavati u upitima [28]. To su:

- **Filter** – uzorak koji filtrira uzorak koji je unesen kao ulazni podatak. Ključna riječ koja se koristi za upotrebu ovog uzorka je FILTER i nalazi se u WHERE djelu upita. Operator procjenjuje istinu ili laž danog izraza i ovisno o tome prikazuje rezultate. Drugim riječima, prikazuje samo one rezultate u kojima je dani izraz istinit [12].
- **Unija** – uzorak koji prikazuje uniju dva uzorka ulaznih podataka. Ključna riječ je UNION i nalazi se u WHERE djelu upita. Ne uklanja duplikate iz rezultata, ako oni postoje [12].
- **Neobavezan uzorak** – neobavezan uzorak ili OPTIONAL operator koristi se u upitima gdje korisnik nije siguran postoji li neka informacija u rješenju ili ne. Ukoliko korisnik definira varijablu koja ne postoji u SELECT djelu upita, upit će dati prazne rezultate ili biti nevažeći. Zbog toga se definira OPTIONAL operator koji dodaje informacije u rezultat upita ukoliko su one dostupne. Dodaje se na kraju WHERE djela upita [28].

Uzorci mogu značajno usporiti ili ubrzati proces vraćanja rezultata upita**.** Tako se na primjer, uzorak OPTIONAL ne preporučuje koristiti jer on jako usporava izvršavanje upita. Sam uzorak ne smanjuje prostor traženja rješenja i SPARQL procesor koji je zadužen za izvršavanje upita mora raditi dodatan posao. Također, poredak varijabli u trojki može značajno utjecati na brzinu izvršavanja upita. Na primjer, ako neki set podataka ima desetak tisuća trojki, no samo deset svojstava za opis tisuće tih resursa, upit u kojem se definira specifična vrijednost na mjestu predikata, a varijable na mjestu subjekta i objekta, izvršiti će se puno brže nego da je specifična vrijednost na mjestu subjekta, a varijable na preostala dva mjesta [27].

Na efikasnost SPARQL upita imaju utjecaj i modifikatori. Dok modifikator ORDER BY može znatno usporiti SPARQL procesor (ukoliko postoji velika količina podataka za sortirati), modifikator LIMIT ga može znatno ubrzati. Također, što je manje varijabli u SELECT izjavi, upit se brže izvršava [27].

2.3.6. RIF – Rule Interchange Format

RIF ili *Rule Interchange Format* je započeo kao grupa W3C-a za kreiranje standarda za razmjenu pravila među različitim sustavima pravila, onima vezanih za područje

semantičkog Web-a, no i za ostala područja u kojima se pravila primjenjuju [11]. W3C je odlučio napraviti standard za razmjenu pravila umjesto osmišljanja novog jezika pravila koji bi odgovarao svim slučajevima u kojima se primjenjuje [12].

Sustavi pravila pripadaju jednoj od tri kategorije: logičko programiranje, logika prvog reda i pravila akcije. Kategorije se jako razlikuju u sintaksi, ali i semantici samih pravila. Zbog toga je grupa W3C-a zadužena za izradu RIF-a odlučila samu razmjenu utemeljiti na takozvanim dijalektima, jezicima koji imaju vrlo specifičnu sintaksu i semantiku te su proširivi (postoji mogućnost dodavanja novih dijalekata) i uniformni (dijalekti su međusobno slični u sintaksi i semantici) [12].

Grupa za razvoj RIF-a se fokusirala na izradu dvije vrste dijalekta: dijalekti bazirani na logici i dijalekti za pravila akcije. Dijalekti bazirani na logici, kao što im i samo ime govori, razvijeni su za jezike koji se temelje na nekoj logici kao što je logika prvog reda ili jezici za logičko programiranje, dok dijalekti za pravila akcije uključuju razvojne sustave pravila kao što su JEss, Drools, JRules i drugi [12].

RIF dijalekti se sastoje od terma i formule. Term uključuje konstante, varijable, članstva, okvire i vanjske termove. Termovi se dijele na jednostavne i osnovne terme. Formule se sastoje od atomskih formula, uvjetnih formula, implikacija, univerzalnih činjenica, grupa i dokumenata [45].

Postoje dva logička dijalekta, RIF – Core i RIF – BLD (*Basic Logic Dialect*) i jedan dijalekt za pravila akcije, RIF – PRD (*Production Rule Dialect)*. Za razvoj ostalih dijalekata oslanjanju se na ostatak zajednice. Da bi olakšali korisnicima izradu tih dijalekata, osmislili su proširivi razvojni okvir pod nazivom *Framework for Logic Dialects* ili RIF – FLD [12].

3. Prolog

Prolog je jedan od najkorištenijih jezika za logičko programiranje. Naziv dolazi iz riječi ***Pro****gramming in* ***Log****ic.* Njegov razvoj počinje u sedamdesetim godinama prošlog stoljeća, a koristi se za razvoj kompleksnih aplikacija, posebno na području umjetne inteligencije. Kao i većina programskih jezika, Prolog je deklarativan jezik, što znači da se programi baziraju na tehnikama logike kako bi formirali valjane zaključke iz dostupnih dokaza. Svaki Prolog program sastoji se od klauzula koje se sastoje od dva elementa: činjenica i pravila. Program je tad pročitan i spremljen. Korisnik nakon toga upisuje niz pitanja (upita) na koje sustav odgovara koristeći činjenice i pravila koja su upisana u program. To su svi elementi Prologa potrebni za logičko programiranje: činjenice, pravila i upiti [30].

Zbog njegove mogućnosti rješavanja kompleksnih problema zaključivanjem i dokazivanjem matematičkih teorema, mnoge aplikacije razvijene programskim jezikom Prolog su na području matematike i logike. No postoji i mnoštvo kompleksnih aplikacija koje nisu: savjetodavni sustavi na području prava, aplikacija za održavanje baze podataka ljudskih gena (Human Genome projekt) na području medicine, aplikacija za automatsko generiranje priče na području video igara, aplikacija za analizu i mjerenje društvenih mreža i mnoge druge. Osim toga, Prolog se koristi i za osnovnu reprezentaciju znanja (KR) semantičkog Web-a, te je jedan od standardnih jezika za razvoj umjetne inteligencije i ekspertnih sustava [30].

3.1. Uvod u Prolog

Prolog program se sastoji od jednog ili više ciljeva [30]. Primjer jednog jednostavnog programa prikazan je ispod:

```
?- write('Hello World'),nl,write('Uvod u Prolog'),nl.
```

Program se sastoji od niza ciljeva koji mora obavezno završavati točkom. Početni dio **?-** se naziva *system prompt* i on je automatski upisan na početku rečenice, korisnik ga ne upisuje ručno [30]. Izvršeni program daje sljedeći rezultat:

```
Hello World
Uvod u Prolog
Yes
```

Program se sastoji od nekoliko dijelova. Prvi dio je **write('Hello World')** i to je ujedno i prvi cilj. Ključna riječ **write** se naziva predikat i to je jedan od mnogih predikata ugrađenih u Prolog. Unutar zagrada i navodnih znakova upisan je željeni tekst, u ovom slučaju **Hello World** koji se ispisan u izvršnoj verziji programa. Drugi cilj, **nl**, dolazi nakon zareza (koji označava logičku konjunkciju) i on je zapravo skraćenica riječi *new line* ili novi redak. To je još jedan od ugrađenih predikata. Nakon toga dolaze preostala dva cilja, **write** i **nl.** Na kraju programa ispisuje se riječ **yes** kako bi program označio uspješno izvršavanje zadatka. Ovakav niz jednog ili više ciljeva se još naziva i upit (engl. *query*) [30].

No Prolog programi se rijetko sastoje samo od upita. Prolog Programi se sastoje od baze podataka u koju se ubacuju klauzule. Tek nakon toga se postavljaju upiti koji daju odgovore na temelju podataka iz baze [30]. Još jedan primjer jednostavnog programa:

```
pas(moki).
macka(felix).
zivotinja(X):- pas(X).
```

Program se sastoji od tri klauzule i svaka završava točkom. Kao što je spomenuto, klauzule mogu biti pravila ili činjenice. Prva klauzula **pas(moki)** je činjenica, a može se interpretirati kao „moki je pas“. Bitno je napomenuti da se nazivi činjenica i njihovih vrijednosti pišu malim slovima. Činjenica se sastoji od predikata. U prethodnom slučaju, predikat je **pas** i on ima jedan argument, **moki.** Argumenti se navode unutar zagrada. Argument **moki** se naziva atom i on je konstanta. Zadnja klauzula programa **zivotinja(X):- pas(X)** je pravilo. Skup znakova **:-** se može interpretirati kao riječ „ako“, a **X** je varijabla koja može predstavljati bilo koju vrijednost. Ovo pravilo može se interpretirati kao „X je životinja, ako je X pas (vrijedi za bilo koji X)“ [30]. Iz toga se lako može zaključiti da je **moki** životinja. Prolog donosi taj zaključak ukoliko upišemo sljedeći upit:

```
?- zivotinja(moki).
yes
```

No, ukoliko upišemo upit:

```
?- zivotinja(felix).
no
```

dobivamo negativan odgovor jer nigdje u programu (bazi Prologa) nije definirano pravilo da je svaka mačka X, također i životinja [30].

3.2. Sintaksa Prologa

U ovom poglavlju opisana je sintaksa osnovnih koncepata Prologa i njegovih struktura podataka. Neke stvari su već spomenute u prethodnom poglavlju, no ovdje je dan detaljniji opis svakog koncepta. Koncepti od kojih se prolog sastoji su [30]:

- Klauzule
- Predikati
- Podatkovni objekti (engl. *data objects*) – atomi, brojevi, varijable
- Strukturirani objekti

3.2.1. Klauzule

Kao što je spomenuto, Prolog program se sastoji od klauzula. Klauzule mogu biti napisane svaka u svojem retku ili jedna do druge, dok god su odvojene točkom i jednim razmakom. Postoje dva tipa klauzula: činjenice i pravila [30]. Činjenice su definirane sljedećom sintaksom:

```
glava.
```

Riječ **glava.** je činjenica koja se sastoji samo od glave klauzule. Činjenicama se mogu dodati i atomi [30].

```
pas.
pas(moki).
majka(ana, ivana).
```

Druga vrsta klauzula, pravila imaju sljedeću sintaksu [30]:

```
glava:-t1,t2,…,tn.
```

gdje je riječ **glava** glava klauzule ili pravila, dok su **t1, t2, …, tn** atomi ili složeni termi i još se nazivaju tijelom pravila. Tijelo pravila specificira skup uvjeta koji moraju biti ostvareni kako bi zaključak predstavljen glavom pravila bio valjan. Skup znakova **:-** je poznat kao vrat pravila i interpretira se kao „ako“. Primjer pravila [30]:

```
divlja_zivotinja(X):- zivotinja(X), divlja(X).
baka(X,Y):- majka(X,Z), roditelj(Z,Y).
```

3.2.2. Predikati

Svaka klauzula koja ima definiran funktor i kratnost se naziva predikat. Predikati mogu imati isti naziv dok god se međusobno razlikuju po kratnosti. Predikati su dodatno objašnjeni pomoću sljedećeg primjera [30]:

```
roditelj(ana)
roditelj(ana, valentino).
roditelj(X,Y):- majka(X,Y).
roditelj(X,Y):- otac(X,Y).
roditelj(X):- sin(Y,X).
majka(marija, ana).
otac(ivan, ana).
sin(valentino, ana).
```

U primjeru postoje dva predikata s istim imenom **roditelj** no svaki ima svoju kratnost. Predikate i njihove kratnosti može se zapisati i kao **roditelj/1** i **roditelj/2**, tj. **nazivPredikata/kratnost** [30].

Predikati se učitavaju u Prolog pomoću ugrađenih predikata **consult/1** i **reconsult/1**. Oba predikata imaju funkciju učitavanja novih predikata u bazu, no međusobno se razlikuju u nekoliko stvari. Ukoliko postoje dva Prolog programa i svaki ima svoje predikate od kojih su neki isti i u prvom i u drugom programu, učitavanjem tih programa putem **consult/1** predikata svi predikati se međusobno spoje u jednom programu i ukoliko su dva predikata ista dolazi do ponavljanja, dakle, isti predikati se ne zamjenjuju već se dvaput ili više javljaju u tom programu. Prilikom učitavanja predikata iz dva programa, najprije putem **consult/1** predikata, a zatim putem **reconsult/1** predikata, predikati iz drugog programa (učitanog pomoću **reconsult/1**) u potpunosti zamjenjuju sve predikate istog naziva u bazi, dakle, neki predikati iz prvog programa nedostaju. Ovakva vrsta učitavanja korisna je prilikom ažuriranja neke verzije programa u kojoj je potrebno obrisati stare linije koda i zamijeniti ih novima [29].

3.2.3. Podatkovni objekti - Termi

Podatkovni objekti se u Prologu nazivaju termi. Postoji nekoliko različitih tipova terma, a svaki je opisan u sljedećim odlomcima.

Brojevi su termi koji omogućuju pisanje integera, tj. cijelih brojeva. Mogu biti pozitivni ili negativni, ovisno o predznaku koji je upisan (+/-). Prolog omogućava dodavanje i decimalnih točaka. Primjeri brojeva su: 123, -23, -.22, +321.22, itd. [30].

Atomi su konstante koje nemaju numeričku vrijednost. Mogu biti pisani na jedan od tri načina [30]:

- Kao niz jednog ili više slova, brojeva i donjih crta, te obavezno moraju početi malim slovom. Na primjer: john, danas_je_Cetvrtak, john_jones, john31, itd. Pogrešni primjeri su: John, 31john, danas-je-cetvrtak, itd.
- Kao niz jednog ili više znakova koji se nalaze unutar jednostrukih navodnika. Na primjer: 'Danas Je Cetvrtak', '32John', 'john-jones', itd.
- Kao niz jednog ili više specijalnih znakova koji može uključivati sljedeće znakove: + - * / > < = & # @. Na primjer: +++, >=, >, +--, itd.

Varijable se u upitu koriste za prikaz terma kojeg treba utvrditi. Na primjer, varijabla X se koristi za atom *pas*, broj 10.3, složeni term ili listu. Sastoji se od niza jednog ili više slova koji uvijek počinje velikim slovom, a može početi i donjom crtom. Primjeri varijabli su: X, Autor, Osoba_A, _1osoba, itd. Varijabla koja počinje donjom crtom naziva se anonimna varijabla. Anonimne varijable se koriste u slučajevima kad vrijednosti nekih varijabli nisu potrebne [30]. Za primjer je napisan upit u kojem korisnik želi pronaći osobu koja se zove John i dobiti samo njegovo prezime:

```
?- osoba(ana,Prezime,_,_,_).
Prezime = horvat
```

3.2.4.Strukture – Složeni termi

Složeni termi ili strukture započinju s atomom koji se naziva funktor. Nakon funktora slijedi niz jednog ili više argumenata koji se nalazu unutar zagrada i odvojeni su zarezima. Opća forma složenih terma je *functor(t1, t2, …, tn).* Primjer jedne strukture je datum, gdje je funktor sam naziv strukture, tj. datum, a njegovi argumenti su dan, mjesec i godina, **datum(29,srpanj,2021)**. Bitno je napomenuti da argumenti mogu biti varijable ili druge strukture. Također, u istom Prolog programu može postojati više struktura s istim imenom, dok god svaka struktura ima različitu kratnost. Kratnost je određena brojem argumenata u složenim termima [30, 31].

Liste se često smatraju posebnom vrstom strukture ili složenog terma, no mogu se tretirati i kao zasebni objekti. Liste imaju neograničen broj argumenata (elemenata) koji se nalaze unutar uglatih zagrada i odvojeni su zarezima. Za razliku od kratnosti kod struktura, broj elemenata u listi ne mora biti prethodno zadan. Elementi liste mogu biti bilo koja vrsta terma. Neki primjeri uključuju: [pas,macka,y,predikat(A,b,c),[el1,el2,el3],x], [[john,31],[marija,28,profesorica]], itd. [31].

3.3. Usklađivanje

Usklađivanje (engl. *matching*) je operacija koja se obavlja na termima. Proces kao ulaz uzima neka dva terma i provjerava jesu li oni jednaki. Ukoliko su termi jednaki, kaže se da je proces uspio (engl. *succeeds*), a ako su različiti, kaže se da proces nije uspio (engl. *fails*). Dva terma su usklađena ukoliko je zadovoljen jedan od tri uvjeta [31]:

- Ako su S i T konstante, međusobno su jednaki samo ako su to isti objekti
- Ako je S varijabla, a T bilo što drugo, međusobno su jednaki i S je instanciran na T, tj. S poprima vrijednost T
- Ako su S i T strukture, međusobno odgovaraju samo ako imaju isti funktor i svi njihovi argumenti su usklađeni

Usklađivanje se može lakše objasniti primjerom. Postoji term **datum(D,M,2021)** i term **datum(D1,srpanj,Y1).** Ta dva terma si međusobno odgovaraju, tj. jednaki su. Jedno od instanciranja koje čini ta dva terma jednako je [31]:

- **D** je instanciran na **D1 → D = D1**
- **M** je instanciran na **srpanj → M = srpanj**
- **Y1** je instanciran na **2021 → Y1 = 2021**

Suprotno, termi **datum(D,M,2020)** i **datum(D1,M1,2021)** nisu usklađeni (jednaki) zbog toga što im se godine ne podudaraju [31].

3.4. Backtracking

Backtracking je proces vraćanja na prethodni cilj i pokušaj da se taj cilj ostvari (engl. *resatisfy*), tj. da cilj bude uspješno proveden. Prolog pokušava riješiti svaki cilj od kojeg se upit sastoji, počevši s lijeva na desno. Ciljevi su povezani zarezom i predstavljaju konjunkciju. Za svaki cilj, Prolog pokušava pronaći činjenicu ili glavu odgovarajućeg pravila koja vrijedi za taj cilj. Ukoliko je iz prvog pokušaja uspješno pronađena istinita činjenica ili glava pravila, Prolog prelazi na zadovoljavanje sljedećeg cilja ukoliko on postoji. Ako cilj nije zadovoljen, Prolog koristi proces backtracking-a i postavlja se na mjesto gdje je zadnje zadovoljio cilj, tj. na činjenicu ili glavu pravila koja je zadnja odgovarala zadovoljenom cilju. Nakon toga, pokušava uskladiti novu činjenicu ili glavu pravila. Ukoliko ne uspije, opet prelazi na novu činjenicu ili glavu nakon što se postavio na prethodno mjesto koje je zadovoljavalo cilj, sve dok ne zadovolji trenutni cilj. Ukratko, Prolog isprobava činjenice i glave pravila sve

dok ne uspije zadovoljiti sve ciljeve koji se nalaze u zadanom upitu. Ukoliko uspije zadovoljiti sve ciljeve, upit je prošao i ispisuje se na kraju **yes,** u suprotnom upit nije uspio i ispisuje se **no** [30].

Backtracking se još može obaviti na način da korisnik umjesto pritiska tipke Enter nakon upisivanja upita, pritisne tipku s znakom točke – zarez (**;**). Na taj način korisnik forsira sustav da sam obavi proces backtracking-a i daje jedno po jedno rješenje [30].

3.5. Operatori i aritmetika

U ovom poglavlju objašnjeni su unarni i binarni predikati te kako ih pretvoriti u operatore, nabrojeni su i opisani operatori koji se koriste u Prologu i objašnjena je upotreba aritmetike.

3.5.1. Operatori

Uobičajena sintaksa predikata već je opisana u prethodnim poglavljima (funktor nakon kojeg slijede argumenti pisani unutar zagrada), no postoji i alternativni način pisanja predikata u Prologu koji neki korisnici mogu smatrati jednostavnijim [30].

Predikati koji se sastoje od dva argumenta nazivaju se binarnim predikatima i mogu se konvertirati u takozvane infiksne operatore. Infiksni operatori pišu se na način da se funktor (naziv predikata) nalazi između dva argumenta i ne koriste se zagrade [30]. Primjer infiksnog operatora:

```
ivan jeOtac marko
```

Predikati koji se sastoje od jednog argumenta još se nazivaju i unarni predikati. Unarni predikati mogu biti konvertirani u prefiksne operatore. Prefiksni operatori imaju funktor na početku i nakon njega slijedi argument, također bez zagrada [30].

```
jePas moki
```

Unarni predikati, osim u prefiksne operatore, mogu biti konvertirani i u postfiksne operatore. Kod postfiksnih operatora prvo se piše argument, a nakon toga dolazi funktor [30].

```
moki jePas
```

Ovakva notacija može olakšati čitanje samog programa. Prolog omogućuje i kombiniranje različitih notacija. Tako je u istoj klauzuli moguće upotrijebiti sve tri notacije operatora i program će biti ispravno izvršen [30].

Svi predikati koje je korisnik sam definirao, a imaju jedan ili dva argumenta, mogu biti konvertirani u operator koristeći ugrađeni predikat **op** [30]. Ovaj predikat se sastoji od tri argumenta i može izgledati ovako:

```
?-op(150,xfy,jeOtac).
```

Prvi argument je prednost operatora. Ona određuje poredak po kojem će se operatori primjenjivati ako neki term ima više od jednog operatora. Prednost je izražena integerom od 0 pa nadalje. Vrijednost broja i prednost su obrnuto proporcionalne, što je manji broj, prednost operatora je veća. Ovaj argument se najčešće koristi za operatore u aritmetici, no više o tome u kasnijim odlomcima [30].

Drugi argument je obično jedan od tri sljedeća atoma [30]:

- **xfy** – označava da je predikat binaran i da se pretvara u infiksni operator
- **fy** – označava da je predikat unaran i da se pretvara u prefiksni operator
- **xf** – označava da je predikat unaran i da se pretvara u postfiksni operator

Slova tih atoma imaju svoje značenje i poredak je bitan. Slovo **f** predstavlja operator, a slova **x** i **y** predstavljaju argumente. Ako se **f** pojavljuje između **x** i **y**, to znači da je operator infiksni, ako je ispred argumenta, znači da je prefiksni, a ako je nakon argumenta, operator je postfiksni. Također, slovo **x** predstavlja argument koji ima veći prioritet (manju brojčanu vrijednost), a slovo **y** predstavlja argument s manjim prioritetom [31].

Treći argument predstavlja naziv predikata koji će se konvertirati u operator [30].

Potrebno je napomenuti da se predikati mogu pretvoriti u operatore na dva načina. Jedan način je već spomenut prije, u *system prompt* (dio za upis upita) se može upisati op predikat, a drugi način je da se op predikat doda na početak programa ili bar prije prve linije koja koristi predikat koji se želi pretvoriti u operator. U ovom slučaju prva dva znaka (?-) ili *prompt* mora obavezno biti dodan na početku. Ovakve linije koda (koje uključuju *prompt* tj. znakove ?-) se nazivaju direktive [30].

Osim korisnički kreiranih predikata koji se mogu konvertirati u operatore, postoje i predefinirani operatori nastali iz ugrađenih predikata. Predefinirani operatori ili relacijski operatori su operatori koji služe za uspoređivanje numeričkih vrijednosti. Relacijskim

operatorima pripadaju *veće od* >, *manje od* <, *jednako* =, *veće ili jednako* >=, *manje ili jednako* =<, *različito* /= i mnogi drugi. Mogu se zapisati na dva načina [30]:

```
X>4 ili >(X,4)
```

3.5.2. Aritmetika

Iako je Prolog uglavnom jezik koji se koristi za simboličke izračune i rijetko se koriste brojčani izračuni, ipak postoji potreba za aritmetikom. Prolog ima predefinirane operatore koji se mogu koristiti za osnovne aritmetičke operacije poput zbrajanja (+), oduzimanja (-), množenja (*), dijeljenja (/) i dijeljenja s ostatkom (mod) [31].

No upotreba tih operatora nije dovoljna da Prolog odgovori na jednostavan upit poput zbrajanja dva broja [31]. Naime, ukoliko se u Prolog *prompt* upiše sljedeći izraz:

```
?- X=1+2
```

on će jednostavno ponovno ispisati **X=1+2** umjesto **X=3**. Da bi korisnik dobio željeni rezultat potrebno je koristiti predefinirani operator **is** [31]. Ovaj operator forsira program da obavi izračun koji je korisnik upisao, a koristi se na sljedeći način:

```
?- X is 1+2
```

U ovom slučaju će korisnik dobiti očekivani rezultat, tj. **X = 3**. Ova procedura, koja se još naziva i ugrađena procedura (engl. *built-in procedure*) koristi se za aritmetičke izraze. Na isti način funkcioniraju svi ostali aritmetički operatori [31].

Ukoliko korisnik želi usporediti vrijednosti neka dva argumenta, ne mora koristiti predikat **is** već jednostavno između dva argumenta stavi željeni operator uspoređivanja, a rezultat uspoređivanja je jednostavan **yes** ili **no**, ovisno o tome vrijedi li usporedba ili ne [31].

Također je potrebno napomenuti da u Prologu postoji nekoliko operatora za izražavanje jednakosti između dva argumenta i nije svejedno koji se koristi. Operator = se koristi za usklađivanje objekata X i Y i može im pridružiti vrijednosti. On ne vrijedi kod računanja [31]. Na primjer:

```
?- 1+2 = 2+1
no

?- 1+A = B+2
A = 2
B = 1
```

Drugi operator jednakosti je **=:=** i on se koristi kod aritmetičkih izračuna i ne može varijablama pridodati nikakve vrijednosti. Dakle, koristeći operator **=:=**, prvi slučaj iz prethodnog primjera bi vrijedio, tj. bio bi istinit [31].

```
?- 1+2 =:= 2+1
yes
```

Treći operator **==** koristi se slučajevima kad korisnik želi provjeriti je li prvi term identični drugom termu. Ovaj operator se može koristiti samo ako su oba argumenta termi. Postoje i operatori nejednakosti za svaki od ova tri operatora jednakosti i funkcioniraju na isti način kao i njihov odgovarajući operator jednakosti [30].

3.6. Unos i ispis

Prolog kao i mnogi drugi programski jezici omogućava unos i ispis terma ili znakova na korisnički ekran ili na korisnikovo računalo. Za unos i ispis koriste se ugrađeni predikati za koje nije omogućena funkcija backtracking-a (backtracking će uvijek biti neuspješan) [30].

3.6.1. Ispis

Glavni ugrađeni predikat za ispis terma je **write/1**. Ovaj predikat ima samo jedan argument koji mora biti valjani Prolog term. Korištenjem ovog predikata za ispis, korisniku se ispisuje upisani term na njegov ekran. Moguće je upisati i atome kao argument, no tada ih je potrebno staviti unutar jednostrukih navodnika. Ukoliko je potrebno te navodnike prikazati u samom ispisu, moguće je koristiti **writeq/1** predikat [30]. Primjeri korištenja predikata za ispis:

```
?- write(26),nl.
26
yes
```

```
?- write('Hello world!'),nl.
Hello World!
yes

?- write('Moki'),nl.
'Moki'
yes
```

Moguće je promijeniti mjesto na koje se podaci ispisuju. Dakle, umjesto ispisa na korisnikov ekran, podatke je moguće ispisati i u zasebni dokument. To je moguće napraviti korištenjem predikata **tell/1.** Ovaj predikat ima jedan argument, a to je atom ili varijabla koja predstavlja naziv dokumenta zajedno s njegovom pripadnom ekstenzijom (npr. „file.txt“). Ukoliko taj dokument ne postoji, Prolog će ga automatski kreirati [30].

3.6.2. Unos

Za unos podataka koristi se ugrađeni predikat **read/1** koji prima jedan argument i on mora biti varijabla. Prilikom korištenja predikata za unos, term koji se unosi je usklađen s varijablom. Ukoliko varijabla nema nikakvu vrijednost, ona poprima vrijednost varijable koja je unešena [30]. Neki primjeri korištenja predikata za unos:

```
?- read(X).
: john
X = john

?- read(X).
: myPredicate(a,b,c)
X = myPredicate(a,b,c)
```

Kao što je slučaj kod ispisa, tako se i kod unosa može promijeniti izvor s kojeg korisnik unosi podatke. Predikat koji mijenja izvor unosa je **see/1.** Kao i prijašnji predikat, ima jedan argument, atom ili varijablu, koja je zapravo naziv dokumenta iz kojeg se žele učitati podaci [30].

3.7. Petlje

Još jedna sličnost s ostalim programskim jezicima je upotreba petlji (engl. *loops*). Petlje omogućuju da se neki skup instrukcija ponavlja određeni broj puta ili dok se ne zadovolji neki uvjet. Iako Prolog nema zasebne funkcije za razne vrste petlji kao što su na primjer *for* petlja ili *do while* petlja, slične mogućnosti mogu se ostvariti ugrađenim predikatima [30].

Jedna od petlji koja je vrlo slična *for* petlji u jezicima poput C#, C++, Python-a i ostalim, u Prologu se naziva **loop. Loop** predikat funkcionira na temelju rekurzije. Primjer programa u kojem se koriti **loop** predikat kako bi se ispisivali brojevi od N pa sve do 1 [30]:

```
loop(0).
loop(N) :- N>0, write('n = '), write(N), nl, M is N-1, loop(M).
```

Osim **loop** predikata za kreiranje petlji, postoji i **repeat** predikat koji se koristi za petlje koje obavljaju određenu funkciju sve dok zadani uvjet nije ispunjen. Vrlo je slična *do while* petlji u ostalim programskim jezicima. Početni ciljevi uvijek uspijevaju i petlja završava tek dok je zadnji cilj ispunjen, tj. istinit. Predikat uspješno radi s backtracking-om i zamjena je za rekurziju koja je opisana u sljedećem poglavlju [30].

3.7.1.Rekurzija

Rekurzija je funkcija koja može pozivati samu sebe sve dok neki cilj nije zadovoljen. U Prologu dolazi do rekurzije dok neki predikat sadrži cilj koji se odnosi na samog sebe. Rekurzija ima dva djela. Prvi dio je činjenica koja zapravo služi kao uvjet koji nakon što je ispunjen zaustavlja rekurziju, a drugi dio je pojednostavljeno pravilo koje poziva samo sebe. Prvi uvjet se provjerava tokom cijelog provođenja rekurzije i ona traje sve dok se taj uvjet ne postane istinit. Bitno je da rekurzivno pravilo nikad ne poziva samog sebe s istim argumentima jer tad rekurzija nikad neće završiti i program će otići u beskonačnu petlju [32].

Primjer kratkog programa koji koristi rekurziju kako bi pronašao pretka:

```
roditelj(john, ana).
roditelj(ana, ivan).
roditelj(ivan, marija).
predak(X,Y) :- roditelj(X,Y).
predak(X,Y) :- roditelj(X,Z), predak(Z,Y).
?- predak(john, ivan).
```

Prolog najprije provjerava što može pronaći o predikatu **predak/2.** Nakon toga pokušava provesti cilj **roditelj(john, ivan)** koji će naravno biti neuspješan jer nigdje nije definiran ovakav predikat. Nakon toga, Prolog pokušava provesti drugu liniju gdje se pojavljuje predikat **predak** i pokušava provesti novi cilj **roditelj(john, Z)**. Prolog pronalazi da je **Z = ana** i pokušava provesti cilj **roditelj(ana, ivan)** koji je uspješno proveden. To znači da cilj **predak(john, ivan)** je uspješan i s time je rekurzija uspješno završila [32].

3.8. Rez i negacija

Iako je proces backtracking-a jedan od osnovnih načina na koji Prolog zadovoljava ciljeve, ponekad njegovo korištenje može dovesti do nepotrebnih ili netočnih rezultata. Da bi korisnik spriječio backtracking i dobio željene rezultate, može se koristiti ugrađeni predikat **cut**, u prijevodu rez [30].

Rez se u Prologu označava uskličnikom (!) i njegovo korištenje može uvelike povećati efikasnost programa i smanjiti vrijeme izvršavanja. Koristan je za slučajeve u kojima se koriste *if then else* izjave. Primjer korištenja reza u programu koji traži najveći broj [31].

```
max(X,Y,Max).
max(X,Y,X):- X>=Y, !.
max(X,Y,Y).
```

Ukratko objašnjeno, Max je broj X ako je X veći ili jednak Y ili je Max Y ako je X manji od Y. U prvom pravilu se može koristiti rez kako bi program završio s provjeravanjem drugog broja ukoliko se uspostavi da je X veći. Na taj način nema ponovne provjere i program se brže izvršava [31].

Osim za prekid backtracking-a, rez se može koristiti i za izražavanje negacije. Obično se za izražavanje negacije koristi ugrađeni predikat **fail**. Rez se može kombinirati s tim predikatom za povećanje efikasnosti samog programa. Jedan jednostavan primjer kojim se iskazuje „Ana voli sve životinje osim zmija“ i upotrebljava rez i predikat **fail** glasi [31]:

```
voli(ana, X):- zmija(X),!,fail;zivotinja(X).
```

Za izražavanje negacije postoji i ugrađeni predikat **not/1.** On predstavlja negaciju kao neuspjeh, isto kao i slučaj s upotrebnom reza i predikata **fail**. Ova vrsta negacije ne

odgovara u potpunosti negaciji u matematičkoj logici pa postoje slučajevi u kojima ju je bolje izbjegavati, sve ovisno o programu koji se piše [31].

3.9. Prolog za semantički Web

Prolog omogućuje pohranu RDF-a, kreiranje upita i zaključivanje sve unutar istog okruženja, što nije tipično za tipične alate semantičkog Web-a. Obično se pohrana RDF-a i kreiranje upita obavlja u posebnim modulima aplikacije. Zbog ovakvog jedinstvenog okruženja koje ima sve tri mogućnosti, Prolog omogućuje efikasno izvršavanje raznih zadataka vezanih uz semantički Web [33].

RDF pruža Prologu stabilni model za reprezentaciju znanja (KR) sa zajedničkom semantikom. Također, daje mogućnost upravljanja web servisima (HTTP protokol), a to se posebno vidi u slučajevima gdje je potrebno generirati dinamičke HTML stranice zajedno s JavaScriptom i JSON serijalizacijom. Za potrebe ovakve vrste programiranja, razvijeno je nekoliko Prolog paketa koji su dostupni za preuzimanje na službenoj stranici Prologa [34].

Jedan od paketa koji Prologu daje mogućnost upotrebe HTTP protokola i kreiranje klijent – server aplikacija je *SWI – Prolog HTTP support* paket. Drugi paket koji je isto tako bitan za područje semantičkog Web-a je SWI – Prolog semweb paket. Ovaj paket omogućuje učitavanje i pohranu RDF podataka te kreiranje upita nad tim podacima. Koristi se uz ClioPatria paket koji je osnova za razvoj aplikacija semantičkog Web-a [34].

Problemi s kojima se programeri susreću prilikom korištenja Prologa za izradu aplikacija semantičkog Web-a su različite pretpostavke svijeta. Naime, Prolog se temelji na pretpostavki zatvorenog svijeta, što znači da ako se neka informacija ne nalazi u bazi, ona je lažna. U suprotnom, semantički Web je temeljen na pretpostavci otvorenog svijeta koja kaže da ako neka činjenica/informacija ne postoji u bazi, tada ona nije ni istinita ni lažna. Zbog toga može doći do raznih konflikata i potrebno je uskladiti pretpostavke kako bi aplikacije ispravno radile [33].

4. Praktični dio – primjer aplikacije semantičkog Web-a

U radu je dosad spomenuto mnogo ideja primjene semantičkog Web-a u stvarnom svijetu od kojih su neke više, a neke manje ostvarive. Diplomski rad govori o spoju Prologa sa semantičkim Web-om pa je tako i logično da se u praktičnom djelu prikaže spoj tih dviju tehnologija.

Aplikacija koja je osmišljena za ovaj diplomski rad zove se Trip Planner, a glavna funkcionalnost joj je, kao što i samo ime govori, planiranje putovanja. Aplikacija podržava samo Hrvatske gradove i zamišljena je kao alat za promociju turizma u Hrvatskoj, te jedan od načina kako spojiti tehnologije semantičkog Web-a s Prologom.

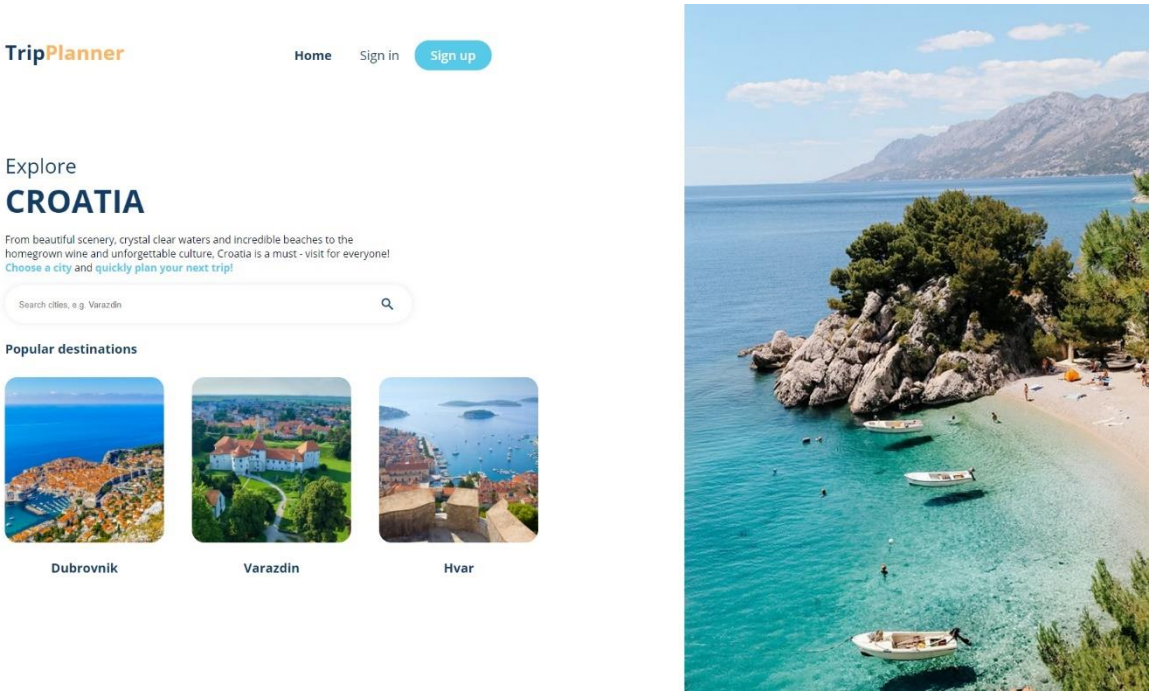
Ova Web aplikacija temelji se na Web tehnologijama HTML, CSS i JavaScript (uz dodatak JQuery-a), programskom jeziku Prologu u kojem je napravljena baza znanja koja pomoću Pengines i Semantic Web paketa pristupa ontologiji kao primarnoj tehnologiji semantičkog Web-a. Detalji implementacije predstavljeni su u kasnijim poglavljima, a najprije su objašnjene sve korištene tehnologije i paketi kako bi prikazani kod bio što jasniji.

4.1. Opis i primjena aplikacije

Glavna svrha aplikacije Trip Planner je jednostavno i brzo planiranje i organiziranje putovanja. Sam koncept planiranja putovanja zamišljen je kao sljedeći niz aktivnosti: korisnik u tražilicu upisuje naziv grada i učitava se lista aktivnosti (znamenitosti, muzeja, sportskih aktivnosti, tura, krstarenja…) koje korisnik u tom gradu može posjetiti.

Korisnik svaku od tih aktivnosti može dodati u svoju listu putovanja koju može kreirati prije ili tijekom dodavanja aktivnosti. Sve liste i aktivnosti u listi se mogu ukloniti i ponovno dodati. Također, korisnik ima mogućnost detaljnijeg prikaza svake od tih aktivnosti u kojima se prikazuju podaci poput opisa aktivnosti, okvirne cijene, recenzije i ostalih informacija (ovisno o vrsti aktivnosti). Korisnik može upravljati svojim listama (kreirati nove ili obrisati postojeće). Za sve aktivnosti osim pretraživanja gradova i pregledavanja detalja aktivnosti potrebno se prije prijaviti u aplikaciju ili ukoliko korisnik još nema račun, registrirati.

Početna stranica aplikacije prikazana je na Slici 5. Tu se vidi glavna tražilica u koju se upisuje naziv grada, a ispod tražilice se nalaze i top 3 destinacije koje korisnik može kliknuti i odmah se prikazuje popis aktivnosti u tim gradovima.



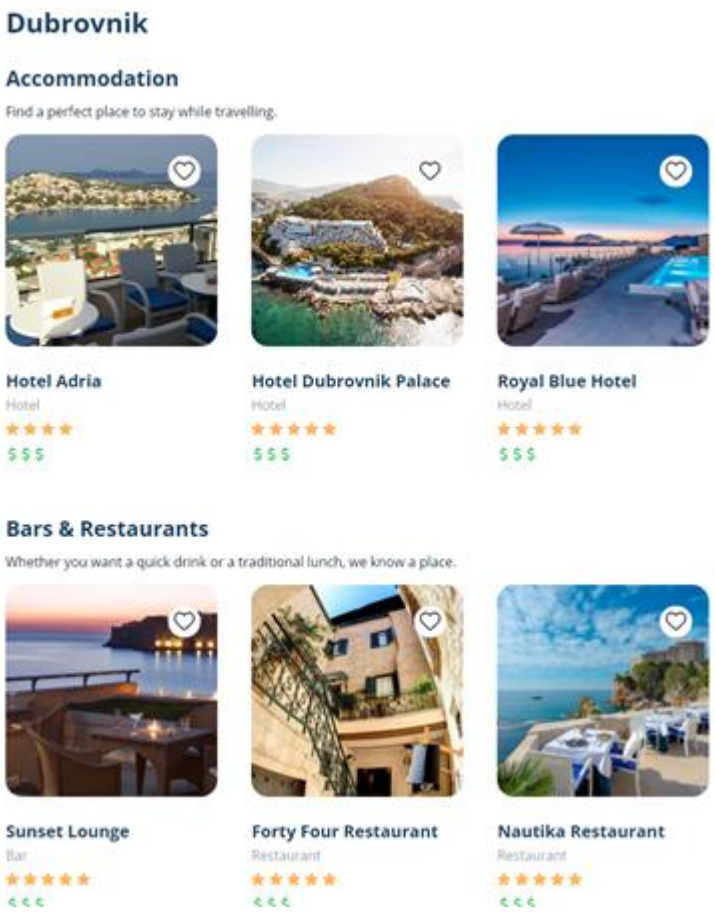
Slika 5. Početna stranica aplikacije

Kada korisnik upiše naziv grada (npr. Dubrovnik) prikazuje se popis aktivnosti. Popis aktivnosti prikazan je na Slici 6. Aktivnosti su grupirane u šest grupa:

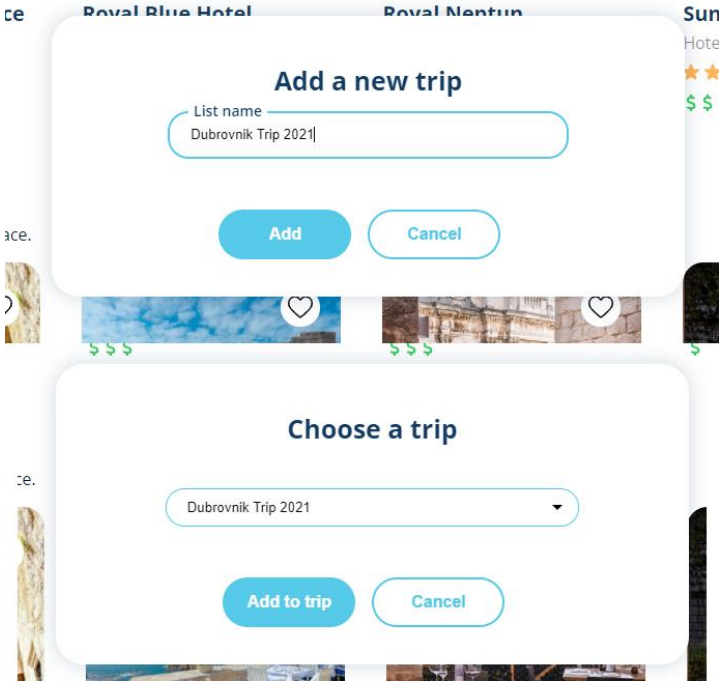
- Smještaj (engl. *Accommodation*) koji se dijeli na hotele i hostele
- Barovi i restorani (engl. *Bars and Restaurants*)
- Znamenitosti i muzeji (engl. *Landmarks and Museums*)
- Ture i krstarenja (engl. *Tours and Cruises*)
- Sport i aktivnosti (engl. *Sports and Activities*)
- Plaže (engl. *Beaches*)

U svakoj od tih grupa nalazi se lista aktivnosti ili smještaja koje korisnik može dodati u svoju listu klikom na gumb s ikonom srca. Ispod slika dan je naziv smještaja/aktivnosti, tip smještaja/aktivnosti koji pripada jednoj od gore spomenutih kategorija, recenzija u obliku broja zvjezdica, te okvirna cijena izražena ikonom dolara ($ - jeftina cijena, $$ - umjerena cijena, $$$ - visoka cijena).

Klikom na gumb za dodavanje aktivnosti (♡) otvara se jedan od skočnih prozora prikazanih na Slici 7., ovisno o tome ima li korisnik već kreiranu listu putovanja ili ne. Ukoliko nema, pojavljuje se prozor za kreiranje nove liste i odabrana aktivnost se automatski dodaje u tu listu.



Slika 5. Lista rezultata



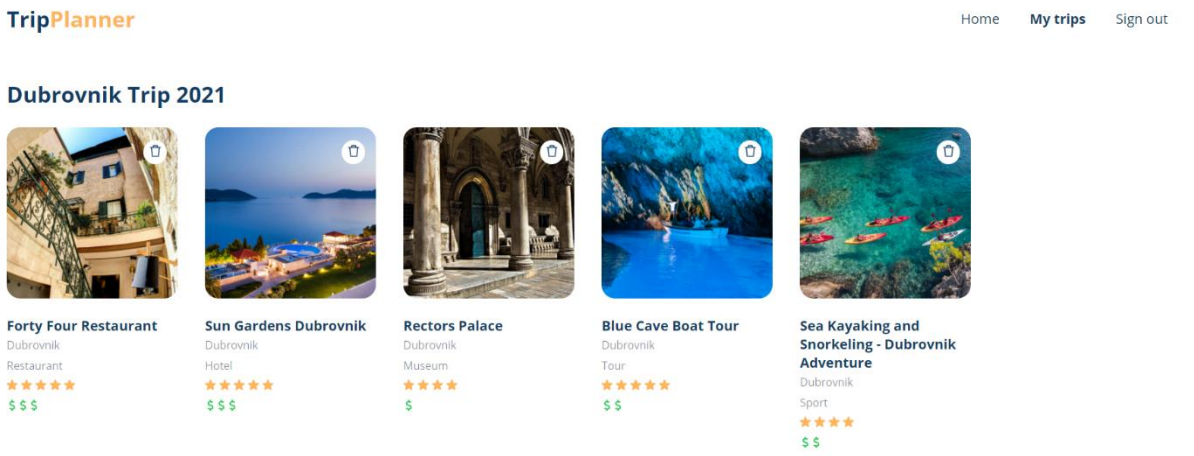
Slika 6. Dodavanje nove liste (gore) i dodavanje nove aktivnosti u listu (dolje)

Lista putovanja (Slika 8.) dostupna je nakon prijave klikom na link „My Trips“ ili „Moja Putovanja“ u navigacijskom izborniku.

Slika 7. Prikaz stranice s listama putovanja

Tu su prikazane sve liste prijavljenog korisnika, kao i gumb za dodavanje nove liste („Add a new trip“), te mogućnost brisanja svake liste pojedinačno (gumb s kanticom za smeće u desnom kutu liste). Brisanjem liste brišu se i sve dodane aktivnosti.

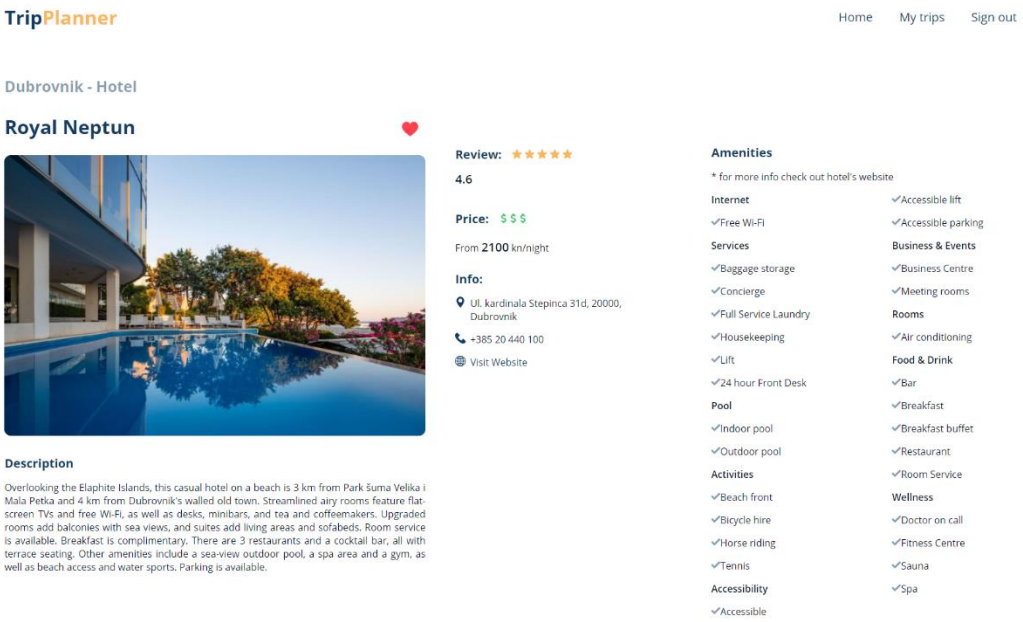
Klikom na listu otvara se nova stranica s popisom svih aktivnosti koje su dodane u tu listu (Slika 9.). Kao i u prethodnom slučaju, svaku aktivnost moguće je i ukloniti iz liste.



Slika 8. Prikaz liste i dodanih aktivnosti

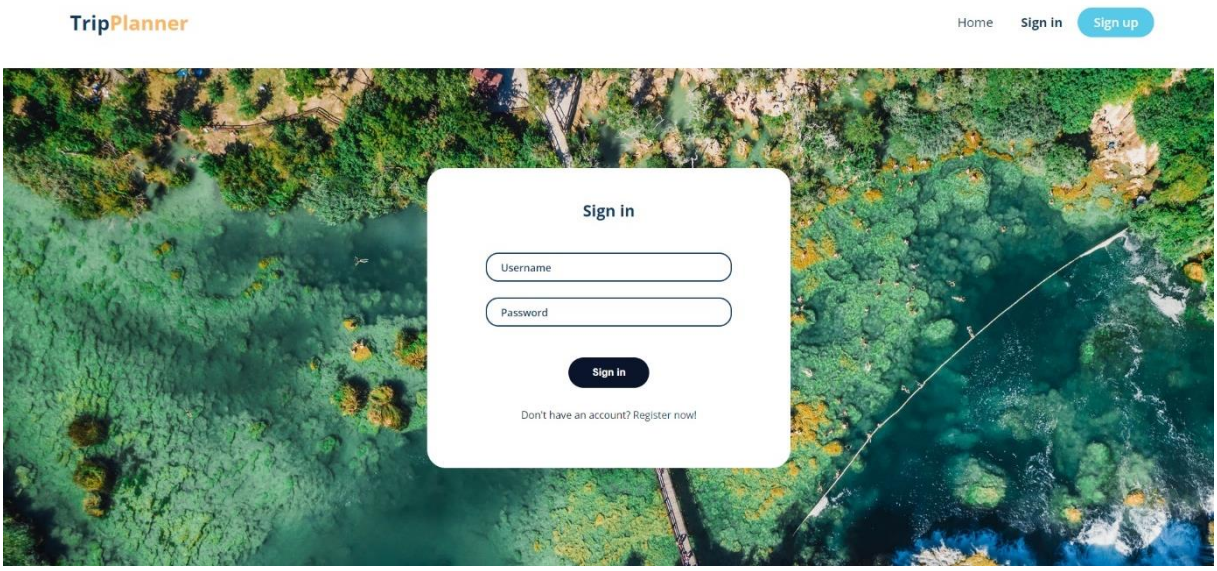
Klikom na bilo koju aktivnost otvara se novi prozor u kojem su prikazani detalji te aktivnosti. Detalji aktivnosti razlikuju se ovisno o tipu aktivnosti i dostupnim informacijama, no uglavnom sve aktivnosti imaju naziv, svoju recenziju, okvirnu cijenu, punu adresu ili naziv mjesta u kojem se nalaze, te web stranicu. Smještajne jedinice, sportovi, ture, krstarenja, plaže, znamenitosti i muzeji imaju svoje opise. Smještajne jedinice imaju i popis sadržaja (engl. *amenities*) koje nude, dok barovi i restorani imaju popis usluga (engl. *services*) koje

(ne) nude. Primjer stranice s detaljima hotela Royal Neptune u Dubrovniku prikazan je na Slici 10.



Slika 9. Prikaz stranice s detaljima o hotelu

Postoje i stranice za prijavu i registraciju korisnika koje se sastoje od jednostavnih formi koje primaju korisničko ime i lozinku. Forme su za potrebe prikaza rada semantičkih tehnologija napravljene vrlo jednostavno te nije potrebna nikakva verifikacija email-a već je dovoljno da korisnik upiše korisničko ime koje ne postoji i potvrdi svoju lozinku prilikom registracije. Nakon toga može odmah prijeći na prijavu. Prijava je prikazana na Slici 11.



Slika 10. Forma za prijavu korisnika

4.2. Korištene tehnologije

Nakon što je opisana primjena i način rada aplikacije, slijedi dio s opisom korištenih tehnologija i implementiranim kodom.

Prije početka razvoja aplikacije istraženi su svi dostupni Prolog paketi koji podupiru tehnologije semantičkog Web-a. Tako su kroz početak rada istraženi paketi poput ClioPatria, SWI – Prolog Semantic Web Library 3.0, SWI – Prolog HTTP Support i na kraju Pengines paket.

4.2.1. ClioPatria i SWI – Prolog HTTP Support

ClioPatria je SWI – Prolog aplikacija koja integrira Prolog pakete za RDF i HTTP servise u već gotov sematički Web server. Ovaj server ima i integrirani upravitelj paketa koji omogućuje dodavanje vlastitih paketa. Temelji se na Prologu i ima jednostavan način transformacije SPARQL upita u Prolog predikate. Također, nudi i optimizaciju SPARQL upita nad Prologom kao i kreiranje kompleksnijih upita [35].

Iako je ClioPatria odličan paket za kreiranje semantičkih aplikacija, on nije korišten u konačnoj implementaciji opisane aplikacije. Primarni razlog zašto nije odabran kao paket za razvoj Trip Planner aplikacije je taj što je kompliciranije napraviti željeno korisničko sučelje.

Naime, korisničko sučelje (HTML stranice uz CSS oblikovanje i JavaScript) potrebno je kreirati u Prologu pomoću SWI – Prolog HTTP Support paketa koji je spomenut u prošlom odlomku. Ovaj paket služi za kreiranje Web servera pomoću Prologa. HTML stranice se u ovom paketu kreiraju na način da se koriste HTTP handleri za kreiranje servera, a HTML i CSS elementi se dodaju pomoću Prolog predikata. Ovaj paket omogućuje i korištenje JavaScripta na način da se uključi u HTML dokument. Primjer jednostavne HTML stranice kreirane u Prologu je dan u odlomku ispod [36].

```
:- use_module(library(http/http_server)).
:- initialization
    http_server([port(8080)]).
:- http_handler(root(.),
                http_redirect(moved, location_by_id(home_page)),
                []).

:- http_handler(root(home), home_page, []).
home_page(_Request) :-
```

```
reply_html_page(
    title('Demo server'),
    [ h1('Hello world!')
    ]).
```

Ovako izgleda jedna jednostavna statička „Hello World“ stranica. Iz linija koda iznad vidljivo je da je za ono što bi u HTML napisali u nekoliko redaka (od kojih je većina automatski generirana), ovdje je potrebno puno više. Između ostalog, potrebno je i puno bolje razumijevanje HTTP protokola nego što je to slučaj kod pisanja običnog HTML-a i pokretanja servera pomoću nekog drugog paketa.

Sam paket je vrlo razvijen i pruža sve moguće opcije za kreiranje Prolog Web servera, no kao što je spomenuto, postaje i poprilično kompleksan za razvoj naprednijih web aplikacija. Upravo zbog te kompleksnosti nije odabran ni jedan od ova dva paketa za razvoj praktičnog djela, no njihovo istraživanje pridonijelo je razvoju konačne ideje za implementaciju aplikacije i njenih funkcionalnosti.

4.2.2. Pengines

Kako ClioPatria nije odgovarala potrebama za implementaciju zamišljene aplikacije, provedeno je daljnje istraživanje o dostupnim Prolog paketima. Zamišljena aplikacija bazira se na Web tehnologijama poput HTML-a, CSS-a i JavaScripta te je bilo potrebno pronaći paket koji može spojiti te tehnologije s Prologom. Tako je pronađen i Pengines paket.

Prolog Engines ili skraćeno Pengines se sastoji od dretvi koje se još nazivaju i *pengines* i često se nalaze na nekom udaljenom Pengine serveru. Omogućuje kreiranje i postavljanje upita u programskom jeziku Prolog. Upiti daju rezultate na temelju podataka koji se nalaze u bazi znanja u obliku n-torki koje ovise o samom upitu. Postoje dva formata zapisa: Prolog sintaksa za Prolog klijente i standardna JSON sintaksa za npr. JavaScript klijente. Također, Pengines paket ima mogućnost kreiranja Prolog upita pomoću JavaScript-a koji je spojen s HTML stranicom [37]. Upravo zbog ove mogućnosti, Pengines je odabran kao primarni paket za izradu semantičke Web aplikacije.

Pengines koristi i prije spomenuti HTTP Support paket, no on je već implementiran u sam paket i nije ga potrebno koristiti za kreiranje dodatnih HTML stranica. Pengines kao i ClioPatria nudi vlastiti server koji se pokreće na predodređenom portu (3030) te izradu vlastitih Web aplikacija. Izrada aplikacije gotova je u samo nekoliko koraka, a detaljniji opis dostupan je u dokumentaciji Pengines paketa [37].

Iako Pengines nije direktno razvijen za izradu semantičkih Web aplikacija, ima mogućnost dodavanja novih paketa. Za rad sa semantičkim tehnologijama odabran je paket opisan u sljedećem poglavlju.

4.2.3. SWI – Prolog Semantic Web Library 3.0

SWI – Prolog Semantic Web Library ili skraćeno *semweb* paket kao jezgru ima učinkovito RDF skladište koje je usko povezano s Prologom. Nudi predikat rdf/3 za postavljanje upita nad RDF bazom uz korištenje različitih indeksa. Također nudi i razne druge pakete za čitanje i pisanje XML/RDF i Turtle sintakse, kao i *persistency* biblioteka koja omogućuje spremanje dinamičkih predikata [38].

Zajedno s ClioPatria paketom pruža efikasnu platformu za razvoj semantičkih Web aplikacija. Osim toga, ima mnoštvo visoko razvijenih biblioteka za upravljanje RDF podacima. Podržava Turtle sintaksu, RDFS, RDFa i SPARQL upite [38].

Iz s*emweb* paketa je primarno korištena biblioteka *rdf_db* i njen rdf/3 predikat. Rdf_db biblioteka pruža skladište za RDF trojke te indeksirano postavljanje upita nad trojkama. Više informacija o samom korištenju paketa i biblioteke dano je u poglavlju s implementacijom koda.

4.2.4. Ontologija i Protégé

Primarna tehnologija semantičkog Web-a koja je korištena za izradu aplikacije je ontologija. Pojam ontologije opisan je u prethodnim poglavljima diplomskog rada i nije ga potrebno ponovno objašnjavati. Alat u kojem je razvijena ontologija zove se Protege.

Protégé je besplatni, open-source alat za razvoj ontologija te razvojni okvir za kreiranje inteligentnih sustava. Protégé ima aktivnu zajednicu korisnika i developera koji neprestano rade na poboljšavanju postojeće dokumentacije, adresiranju problema koji se pojavljuju te na izradi novih dodataka za alat. Podržava W3C standarde kao što su OWL2 i RDF specifikacije [39].

4.3. Implementacija aplikacije

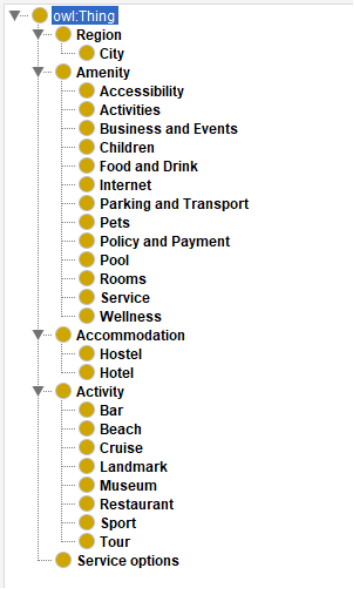
Opis i funkcionalnosti aplikacije dani su u poglavlju 4.1., a kako su te funkcionalnosti realizirane prikazano je u ovom djelu rada. Sama implementacija aplikacije provedena je u nekoliko koraka od kojih će svaki biti detaljno opisan u posebnom poglavlju. Koraci implementacije su sljedeći:

1. Izrada ontologije
2. Kreiranje SPARQL upita nad ontologijom i njihovo prevođenje u RDF upite u Prologu pomoću semweb paketa
3. Kreiranje Web servera i povezivanje s Prolog bazom pomoću Pengines paketa
4. Kreiranje JavaScript skripta za izvršavanje Pengines upita i njihov prikaz na Web stranici

Za implementaciju su odabrani alati Visual Studio Code koji osim što podržava HTML, CSS, JavaScript, jQuery, podržava i Prolog, te alat Protege za izradu ontologije.

4.3.1. Kreiranje ontologije

Izrada ontologije prvi je korak u izradi aplikacije. Prije nego što se uopće počne s kodiranjem i izradom Web aplikacije, potrebno je kreirati ontologiju. Tema ontologije je putovanje kroz Hrvatsku. Na temelju te ideje razvijene su sve potrebne klase, objektna i podatkovna svojstva koja pomažu u opisu putovanja. Prikaz klasa dan je na sljedećoj slici.

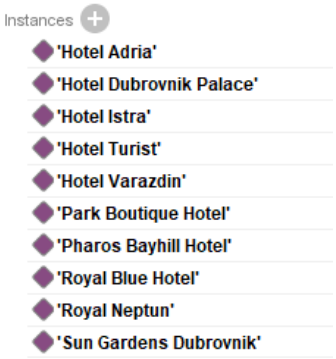


Slika 11. Klase ontologije

Ontologija se ukupno sastoji od pet glavnih klasa: Regija (engl. *Region*), Sadržaj (engl. *Amenity*), Smještaj (engl. *Accommodation*), Aktivnost (engl. *Activity*) i Usluge (engl. *Service Options*). Svaka od tih klasa ima svoje potklase. Tako se je potklasa regije, grad (engl. City), potklase smještaja su hoteli (engl. *Hotel*) i hostel (engl. *Hostel*), potklase aktivnosti su bar (engl. *Bar*), restoran (engl. *Restaurant*), plaža (engl. *Beach*) i ostali. Svakoj potklasi dodana je anotacija rdfs:label tipa xsd:string kako bi SPARQL upiti (pa čak i Prolog upiti) bili što čitljiviji, no više o tome kasnije. Također je bitno napomenuti da su sve klase

međusobno razdvojene jer bi u protivnom prilikom zaključivanja došlo do greške zbog toga što Protege bez tog svojstva ne shvaća da se klase međusobno razlikuju.

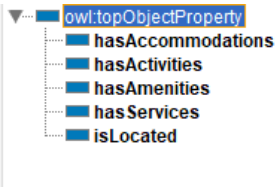
Klase i njihove potklase predstavljaju glavne kategorije rezultata pretraživanja. Sami rezultati pretraživanja su zapravo individue koje su dodijeljene svakoj klasi. Primjer individua dodijeljenih klasi Hotel prikazan je na Slici 14.



Slika 12. Individue klase Hotel

Kao što je i slučaj kod klasa i potklasa, svakoj individui dodana je anotacija rdfs:label te su sve međusobno razdvojene. Osim anotacije rdfs:label dodana je i anotacija schema:image pomoću koje je moguće dodati slikovni prikaz individue. Ova funkcionalnost korištena je za sve individue koje predstavljaju smještaj ili neku aktivnost.

Individuama su dodana objektna i podatkovna svojstva koja ih povezuju s ostalim individuama. Popis objektnih svojstava dan je na Slici 14.



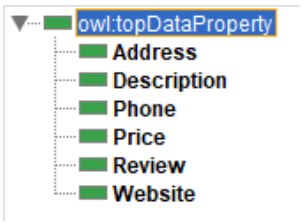
Slika 13. Popis objektnih svojstava

Ontologija se dakle sastoji od pet objektnih svojstava i svako objektno svojstvo ima svoju domenu i doseg:

- hasAccommodations – ima smještajne jedinice
  - Domena: Grad
  - Doseg: Smještaj
- hasActivities – ima aktivnosti
  - Domena: Grad

    - Doseg: Aktivnost
- hasAmenities – ima sadržaje
    - Domena: Smještaj
    - Doseg: Sadržaj
- hasServices – pruža usluge
    - Domena: Bar ili Restoran
    - Doseg: Usluga
- isLocated – nalazi se
    - Domena: Smještaj ili Aktivnost
    - Doseg: Grad

Osim objektnih svojstava, individue imaju i svoja podatkovna svojstva. Podatkovna svojstva prikazana su na Slici 15, a njihovi opisi, domene i dosezi nalaze se u odlomku ispod.



Slika 14. Podatkovna svojstva

Postoji šest podatkovnih svojstva i ona daju osnovne informacije o svakoj aktivnosti i smještaju. Neke aktivnosti/smještaji nemaju sve informacije (neke nisu dostupne, a neke su nepotrebne). Podatkovna svojstva dodana su svakoj individui zasebno i ovise o vrsti aktivnosti (npr. plažama nije potreban broj telefona, a neke nemaju ni određenu adresu).

- Adresa (engl. *Address*)
    - Domena: Smještaj ili Aktivnost
    - Doseg: xsd:string
- Opis (engl. *Description*)
    - Domena: Smještaj ili Aktivnost
    - Doseg: xsd:string
- Telefon (engl. *Phone*)
    - Domena: Smještaj ili Aktivnost
    - Doseg: xsd:string
- Cijena (engl. *Price*)
    - Domena: Smještaj ili Aktivnost

    - Doseg: xsd:int
- Recenzija (engl. *Review*)
    - Domena: Smještaj ili Aktivnost
    - Doseg: xsd:double
- Web stranica (engl. *Website*)
    - Domena: Smještaj ili Aktivnost
    - Doseg: xsd:string

Dodavanjem svih klasa, potklasa, individua i njihovih svojstava, ontologija je završena. Dodano je nekoliko testnih podataka kako bi se aplikacija mogla isprobati. Sve informacije o smještajnim jedinicama i aktivnostima poput recenzija, prosječnih cijena, opisa, usluga i sadržaja koje nude preuzete su s Google-a i ručno su unešene u ontologiju. Sama ontologija se može doraditi na još razne načine i popuniti novim informacijama, no u svrhu prikaza rada semantičke Web aplikacije s Prologom, informacija je sasvim dovoljno.

4.3.2. Prolog baza znanja i RDF upiti

Nakon kreiranja ontologije, potrebno je izraditi bazu znanja u Prologu koristeći *semweb* paket te kreirati nekoliko RDF upita. Prije početka, kreirana je datoteka *travelapp.pl* i dodan je *semweb* paket pomoću sljedeće naredbe:

```
:- use_module(library(semweb/rdf_db)).
```

Nakon toga je potrebno u Prolog datoteku dodati sve potrebne upite koji će poslužiti pri pretraživanju aktivnosti u gradovima i prikupljanju informacija o tim aktivnostima. No prije nego što je započeto pisanje upita u Prologu, napravljeno je nekoliko testnih SPARQL upita u Protege-u koji se prevode u RDF upite kako bi se što lakše shvatili.

Primjer jednog takvog upita je SPARQL upit koji prikuplja informacije o aktivnostima koje se nalaze u određenom gradu zajedno s njihovom kategorijom (natklasom kojoj pripadaju).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX t:
<http://www.semanticweb.org/sara/ontologies/2021/7/croatia_travel#>

SELECT ?activity ?type ?city
```

```
        WHERE { ?type rdfs:subClassOf ?t .

        ?activity rdf:type ?type .

        ?activity t:isLocated ?city .

        FILTER (?t IN (t:Activity, t:Accommodation ) )

}
```

Iz upita vidimo da su najprije dodani prefiksi (osnovni prefiksi poput rdf, owl, rdfs i xsd, te dodatni prefiks t koji označava ontologiju). Nakon toga odabiru se tri varijable: ?activity, ?type i ?city. ?type je potklasa klase t koja zapravo može pripadati klasama Activity ili Accommodation, ?activity pripada tipu ?type i ima objektno svojstvo isLocated koje označava u kojem se gradu ta aktivnost nalazi. Rezultati upita prikazani su na Slici 16.

activity	type	city
Royal Neptun	Hotel	Dubrovnik
Royal Blue Hotel	Hotel	Dubrovnik
Hotel Dubrovnik Palace	Hotel	Dubrovnik
Pharos Bayhill Hotel	Hotel	Hvar
Sveti Jakov Beach	Beach	Dubrovnik
MliniBeach	Beach	Hvar
Sulic Beach	Beach	Dubrovnik
Bellevue Beach	Beach	Dubrovnik
Korzo Varazdin	Landmark	Varazdin
Tvrdava Fortica	Landmark	Hvar
Stradun	Landmark	Dubrovnik
Old Town	Landmark	Dubrovnik
Walls of Dubrovnik	Landmark	Dubrovnik
Lovrijenac	Landmark	Dubrovnik

Slika 15. Rezultati SPARQL upita u Protege-u

Da bi ovaj upit mogli provesti u Prologu, potrebno ga je malo preformulirati. Naime, ontologija je spremljena u RDF/XML obliku i da bi se učitala u Prolog potrebno je upisati sljedeći upit:

```
?- rdf_load('putanja/do/ontologije/croatia_travel.owl').
```

Učitavanje ontologije neće biti dovoljno. Kao što je i u SPARQL upitu bilo potrebno dodati prefiks same ontologije, isti postupak potrebno je obaviti i u Prologu. Neki osnovni prefiksi već su registrirani no prefiks ontologije i schema.org prefikse potrebno je dodati ručno. Prefiksi se dodaju upisivanjem sljedećeg upita:

```
?-
rdf_register_prefix(t,'http://www.semanticweb.org/sara/ontologies/2021/7/croatia_travel#').

?- rdf_register_prefix(schema,'http://schema.org/').
```

Nakon što je učitana ontologija i dodani svi potrebni prefiksi, može se krenuti s kreiranjem RDF upita nad ontologijom. Primarni predikat koji je korišten za kreiranje upita je rdf/3. Ovaj predikat ima tri varijable koje odgovaraju RDF trojki (subjekt, predikat i objekt). SPARQL upiti se također sastoje od tri dijela te se lako mogu transformirati u RDF upite. Primjer transformacije prijašnjeg SPARQL upita u RDF upit glasi:

```
get_city_activity(ActivityName, TypeName, CityName):- (

rdf(Type, rdfs:subClassOf, t:'Activity');

rdf(Type, rdfs:subClassOf, t:'Accommodation')),

rdf(Activity, rdf:type, Type),

rdf(Type, rdfs:label,
literal(type('http://www.w3.org/2001/XMLSchema#string',TypeName))),

rdf(Activity, rdfs:label,
literal(type('http://www.w3.org/2001/XMLSchema#string',ActivityName))),

rdf(City, rdfs:label,
literal(type('http://www.w3.org/2001/XMLSchema#string',CityName))),

rdf(Activity, t:'isLocated', City).
```

Ukratko objašnjeno, najprije je kreiran naziv upita kao naziv predikata (get_city_activity). Nakon njega definirana su tri argumenta: ActivityName, TypeName i CityName. Svaki od ta tri argumenta predstavlja nazive (potklasa) koje su definirane pomoću anotacije rdf:label. Kao što je u SPARQL upitu tip aktivnosti bio definiran kao ?*type rdfs:subClassOf ?t*, u Prologu je ovaj dio definiran kao *rdf(Type, rdfs:subClassOf, t:'Activity')*. Dakle, Prolog definicija tipa aktivnosti i SPARQL definicija su skoro pa jednake. Obje su definirane kao trojke koje imaju subjekt, predikat i objekt. Na isti način definirane su i preostale dvije varijable. U Prologu je dodan i dio koji očitava rdf:label kako bi rezultati upita bili „ljepši“ (bez definicije literala u obliku URL-a). Rezultati Prolog upita za aktivnosti u gradu Varaždinu prikazani su na Slici 17.

```
3 ?- get_city_activity(Activity, Type, 'Varazdin').
Activity = 'Mea Culpa Club and Cafe',
Type = 'Bar' ;
Activity = 'Medina Škrinja',
Type = 'Bar' ;
Activity = 'My Way',
Type = 'Bar' .
```

Slika 16. Dio rezultata provedenog upita

Provođenje upita u Prologu ima omogućen backtracking pa je potrebno pritisnuti znak točka – zarez kako bi dobili određen broj upita.

Na ovaj način napravljeni su svi upiti koji se provode nad Prolog bazom. Da bi se rezultati tih upita mogli koristiti u Web aplikaciji potrebno je Prolog bazu spojiti s Web aplikacijom.

4.3.3.Kreiranje Pengines Web servera

Spajanje Prologa s Web aplikacijom moguće je uz Pengines paket koji pruža i vlastiti Web server. Pengines paket potrebno je najprije preuzeti sa službene Web stranice [47] i ekstrahirati ga u zasebnu mapu. Pengines dolazi s nekoliko vlastitih aplikacija poput SWISH-a i Scratchpad-a kao online Prolog compilera, te primjera Web aplikacije Genealogist. Nova aplikacija kreira se jednostavnim dodavanjem nove mape i *app.pl* u koju je potrebno dodati module kako bi aplikacija bila vidljiva u Pengines admin panelu. Moduli su kopirani iz Genealogist aplikacije i nazivi su prilagođeni novokreiranoj aplikaciji. Pengines admin panelu može se pristupiti na adresi: https://pengines.swi-prolog.org/docs/index.html klikom na Admin link u navigaciji.

Također, potrebno je dodati i modul koji ukazuje na poziciju *app.pl* datoteke u datoteku *load.pl* koja se nalazi u korijenskom direktoriju Pengines-a. Nakon što su dodani svi moduli, Pengines Web server pokreće se klikom na datoteku *run.pl* koja se otvara putem SWI – Prolog-a. Server je pokrenut na lokalnom računalu uz port 3030. Aplikaciji koja je izrađena može se pristupiti pomoću sljedećeg linka: http://localhost:3030/apps/travelapp/index.html.

Nakon što je Web server pokrenut, potrebno ga je spojiti s Prolog bazom. Za to je potrebno u glavnu Prolog datoteku dodati nekoliko linija koda [37]:

```
:- use_module(pengine_sandbox:library(pengines)).
:- use_module(library(semweb/rdf_db)).
:- use_module(pengine_sandbox:library(semweb/rdf_db)).
:- use_module(library(sandbox)).
sandbox:safe_primitive(rdf_db:rdf(_,_,_)).
```

Pengines je korišten u *sandbox* načinu rada zbog sigurnosti, no to se može i isključiti. Za potrebe izrade praktičnog djela nije bilo potrebno isključivati ovaj način rada pa su samo uključene dodatne biblioteke (semweb/rdf_db) koje će Pengines smatrati sigurnima za korištenje.

Osim u Prolog datoteku, potrebno je uključiti Pengines i u svaku HTML stranicu. Pengines se u HTML kod uključuje pomoću <script> oznake i preporuča se dodati na kraj

<body> dijela stranice (veća optimizacija, manje šanse za pojavom neke greške ukoliko neki element još nije učitan). Linija koda za dodavanje Pengines u HTML stranicu:

```
<script src="/pengine/pengines.js"></script>
```

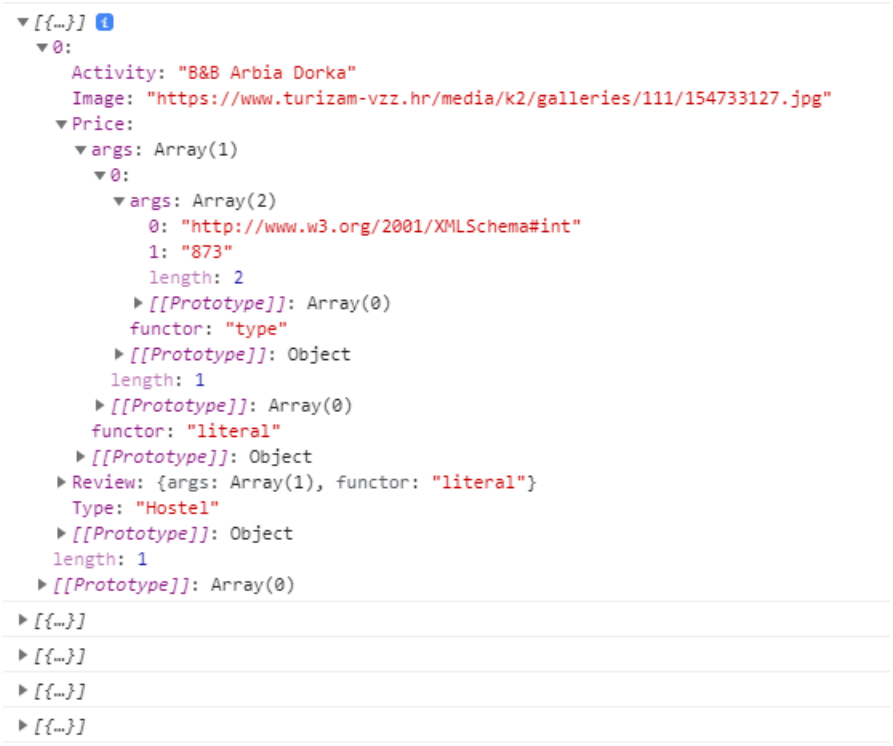
4.3.4. Pozivanje upita u JavaScriptu

Glavni razlog zašto je odabran Pengines za posluživanje Web servera je taj što dobro surađuje s JavaScriptom. Tamo se odvija i provođenje upita i dobavljaju se rezultati u JSON obliku. Glavna funkcija za postavljanje upita nad Prolog bazom glasi [37]:

```
function ask(query){
    pengine = new Pengine({
        application: 'travelapp',
        ask: query,
        destroy: true,
        onsuccess: function() {
            console.log(this.data);
            showResults(this.data);
            if (this.more) {
                pengine.next();
            } else {
                console.log("No more solutions");
            }
        },
        onfailure: function() {
            console.log("Failure")
        },
        onerror: function() {
            console.log("Error: " + this.data);
        },
        ondestroy: function(){
            console.log("Pengine destroyed!");
```

```
        pengine = null;

} });}
```

Prije pozivanja *ask(query)* funkcije u JS-u (JavaScript-u), potrebno je kreirati pengine i postaviti ga na null vrijednost. Ova funkcija prisutna je u svakoj JS datoteci i one su uključene u HTML stranice. Dakle, nakon što se pozove funkcija *ask(query)* kreira se novi Pengine za specifičnu aplikaciju (u ovom slučaju travelapp) i postavlja se upit koji je sadržan u varijabli query. Zbog toga što je *destroy* postavljen na *false*, Pengine se na završetku upita uništava i ukoliko je upit uspješno proveden prikazuju se rezultati pomoću *showResults(this.data)* funkcije ili se ispisuju u konzolu. Rezultati provedbe upita *get_activity_type(ActivityName, TypeName, CityName, Image, Review, Price)* su prikazani u JSON formatu (Slika 18.).



Slika 17. Prikaz rezultata u JSON formatu (konzola)

Funkcija *showResults(this.data)* služi za manipulaciju dobivenim JSON podacima i prikazuje te rezultate na HTML stranici. Prikaz rezultata pretraživanja dan je u poglavlju s opisom i primjenom aplikacije.

Potrebno je napomenuti da je nakon svake promjene ontologije ili baze znanja potrebno napraviti *consult* i ponovno pokrenuti Pengines server kako bi promjene bile vidljive.

Na isti način, dakle pomoću *ask(query)* funkcije kreirane su sve JavaScript skripte i pribavljene informacije o raznim aktivnostima. Osim pretraživanja aktivnosti po gradovima, realizirana je i aktivnost dodavanja i brisanja aktivnosti i lista koja se poziva na isti način kao i svaki upit dosad.

4.3.5. Persistency biblioteka

Da bi se neki upit pozvao u JavaScriptu i izvršio na Web serveru, potrebno ga je najprije definirati u Prologu i proglasiti sigurnim za rad u *sandbox* načinu rada. Upiti su u Prologu definirani kao predikati i svaki ima svoj naziv. Primjer jednog takvog predikata već je prije opisan.

Takve predikate (koji rade s ontologijom i koriste rdf_db biblioteku) nije potrebno posebno definirati, no aplikacija također radi i s drugim dinamičkim predikatima koji upravljaju s korisnicima, njihovim listama i dodanim aktivnostima. Aplikacija ne koristi SQL bazu za pohranu takvih podataka, već je sve pohranjeno u Prolog bazi znanja. Jedna od karakteristika dinamičkih predikata u Prologu je ta što se nakon zatvaranja Prolog-a svaki term pohranjen dinamičkim predikatom briše. Dakle, ukoliko programer odluči pomoću predikata *assert* dodati novi term *user('sara', 'lozinka')*, term će se nakon zatvaranja Prologa obrisati.

Da bi se to spriječilo i da bi kreirana nova baza znanja čuvala takve termove, potrebno je koristiti *persistency* biblioteku. Deklariranje predikata kao [40]:

```
:- persistent
      user(name:atom, pass:atom, role:oneof([regular,administrator])),
```

proširuje term *user* u četiri nova predikata. Jedan od njih je upisani predikat *user/3*, a na druga tri dodaju se prefiksi *assert_*, *retract_* i *retractall_* [40].

Da bi se ti predikati mogli koristiti prilikom pozivanja upita u JavaScript-u, potrebno ih je deklarirati sigurnim za okruženje. Također, ti podaci se moraju i nekamo spremati. Datoteka spremanja određuje se pozivanjem sljedeće linije koda [46]:

```
:- initialization(db_attach('userdb.journal', [])).
```

Datoteci je moguće dati bilo koje ime i bilo koji nastavak (obično su to nastavci .journal ili .pl) i ona služi kao baza podataka.

Primjer predikata za registraciju i brisanje korisnika dan je na sljedećoj stranici:

```
register_user(Username, Password) :- not(user(Username, _, _)) -> assert_user(Username, Password, regular), pengine_output("Registration successful!"); pengine_output("Username already exists!").

unregister_user(Username, Password) :- user(Username,_,_) -> retract_user(Username, Password, regular), pengine_output("Unregistration successful!"); pengine_output("User doesn't exist.").
```

Na ovaj način realizirane su sve ostale funkcije za rad aplikacije. Cijeli kod, kao i upute za pokretanje aplikacije, dostupni su na GitHub repozitoriju pod nazivom TripPlanner. [1]

[1] https://github.com/skisic/TripPlanner

5. Zaključak

Ideja semantičkog Web-a je potpuna kooperacija ljudi i računala. Iako je prošlo već dvadesetak godina od pojave ideje semantičkog Web-a, zamišljeni Web u kojem agenti samostalno obavljaju zadatke i dolaze do jasnih zaključaka još nije ostvaren. U početku je ideja probudila znatiželju u mnogim znanstvenicima što je pridonijelo i razvoju mnogih semantičkih tehnologija kao što su RDF, RDFS, OWL, SPARQL, ontologije, Linked Data i mnoge druge. Neke od tih tehnologija postale su i W3C standardi. Posljednjih nekoliko godina, razvoj semantičkog Web-a je stagnirao, a neki ga čak smatraju i „mrtvim“. U jednom trenutku, ideja semantičkog Web-a ponovno je zaživjela napretkom tražilica, te pojavom SEO-a, no ne u dovoljnoj mjeri da se početni koncept koji je zamislio Tim Berners - Lee i ostvari.

Iako se semantički Web danas i ne koristi (bar ne na način kako je zamišljeno), njegov razvoj pridonio je razvoju velikog broja novih tehnologija kao što su API-ji, umjetna inteligencija i strojno učenje. Ove tehnologije uspijevaju tamo gdje je semantički Web zakazao, a glavni razlog tome je što neki smatraju da su semantičke tehnologije presložene za korištenje. Napravljene su od strane akademika za akademike, dok je semantički Web trebao biti za sve ljude [41]. Umjesto rekonstrukcije svog sadržaja na Web-u kako bi on bio čitljiv računalima, nove tehnologije (poput umjetne inteligencije i strojnog učenja) više se fokusiraju na način kako povezati podatke na način kako su napisani. Dakle, ljudi se ne prilagođavaju računalima, što je zapravo potrebno prilikom kreiranja sadržaja za semantički Web, već ljudi uče računala kako interpretirati podatke koji već postoje, na način kako su i postavljeni na Web.

Što se tiče Prologa i semantičkog Web-a, Prolog kao programski alat nudi pohranu RDF-a, kreiranje upita nad tim podacima kao i zaključivanje, sve unutar istog okruženja. To je jedna od prednosti Prolog-a nad ostalim alatima koji su napravljeni za semantičke tehnologije. Zbog toga što Prolog ima razvijene pakete za rad sa semantičkim tehnologijama i moguće je direktno (bez razvoja dodatnih algoritama) raditi sa semantičkim podacima, brže obavlja potrebne zadatke [33].

Razvijena aplikacija prikazuje osnovnu primjenu semantičkih Web tehnologija i potpuno je proširiva. Ovisna je o razvijenoj ontologiji, dakle, što se više ontologija proširi, to aplikacija nudi više informacija o samim gradovima i aktivnostima. Trenutno korisnik može samo pročitati informacije o smještaju ili nekoj aktivnosti, no jedna od ideja za proširenje je ta da se korisniku direktno sa stranice omogući rezervacija odgovarajućeg smještaja, s odabirom sobe i prikazom cijene. Za to bi trebalo dodatno proširiti ontologiju što može biti

popriličnо dugotrajan proces. Druga mogućnost je korištenje razvijenih API-ja za prikupljanje takvih podataka što je puno jednostavnije i brže, no nije u duhu semantičkog Web-a. Ovdje je također vidljiva ta nedovoljna iskorištenost semantičkog Web-a i kako ga druge tehnologije danas mogu lako zamijeniti. Također, Web aplikacija nema nikakvih sigurnosnih mjera, pošto je baza podataka s korisničkim računima napravljena u Prologu bez ikakvog kodiranja lozinki, pa se može poraditi i na tome.

Semantičke tehnologije i nisu toliko kompleksne za korištenje, no potrebno je neko vrijeme da se shvate, što je danas zapravo nedostatak. Uvijek se biraju jednostavniji i brži načini za izvršavanje takvih zadataka, a semantičke tehnologije su u tome zakazale. Iako ideja semantičkog Web-a nije potpuno nestala, nije ni evoluirala pa je zamijenjena novijim tehnologijama. Još jedan nedostatak je povreda privatnosti. Informacije koje korisnik stavlja na Web dostupne su svima i to povećava rizik korištenja tih informacija u pogrešne svrhe. Tu dolazi do pojave politike privatnosti. No osim nedostataka, semantički Web ima i mnoge prednosti. Znanja sadržana na Web-u dostupna su većem broju korisnika i lakše se dijele, korisnici lakše pristupaju traženim informacijama, stranice koje koriste semantičke oznake vidljivije su u rezultatima pretraživanja čime se može povećati i sam promet te stranice pa tako i neke organizacije ili tvrtke. Semantički Web ima mnogo prednosti, ali i nedostataka te bi se razvojem tehnologije ovo zanimljivo područje svakako moglo ponovno početi razvijati i nadograđivati.

Popis literature

[1] „What is the world wide web?“ [Na Internetu]. *BBC*. Dostupno: https://www.bbc.co.uk/bitesize/topics/zkcqn39/articles/z2nbgk7 [pristupano: 08.04.2021.]

[2] „World Wide Web (WWW) | History, Definition & Facts“ [Na Internetu]. *Britannica*. Dostupno: https://www.britannica.com/topic/World-Wide-Web [pristupano: 08.04.2021.]

[3] „Internet Basics: What is the Internet?“ [Na Internetu]. *GCFGlobal.* Dostupno: https://edu.gcfglobal.org/en/internetbasics/what-is-the-internet/1/ [pristupano: 08.04.2021.]

[4] „HTML Introduction“ [Na Internetu]. *W3Schools.com.* Dostupno: https://www.w3schools.com/html/html_intro.asp [pristupano: 13.04.2021.]

[5] „What is a URL?“ [Na Internetu] *MDN Web Docs.* Dostupno: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_URL [pristupano: 13.04.2021.]

[6] V. Madurai, „Web Evolution from 1.0 to 3.0“ (17.02.2018.) [Na Internetu]. *Medium*. Dostupno: https://medium.com/@vivekmadurai/web-evolution-from-1-0-to-3-0-e84f2c06739 [pristupano: 13.04.2021.)

[7] T. Berners – Lee, J. Hendler, O. Lassila, „The Semantic Web“ (05.2001.) [Na Internetu]. *Scientific American.* Dostupno: https://www.jstor.org/stable/10.2307/26059207 [pristupano: 13.04.2021.]

[8] S. Aghaei, “Evolution of the world wide web : From web 1.0 to web 4.0,” *International Journal of Web & Semantic Technology*, izd. 3, br. 1, str. 1–10, 2012 [Na Internetu]. Dostupno: http://airccse.org/journal/ijwest/papers/3112ijwest01.pdf [pristupano: 13.04.2021.]

[9] Werner Vermaak, „What is Web 3.0?“ (08.02.2021.) [Na Internetu]. *Alexandria.* Dostupno: https://coinmarketcap.com/alexandria/article/what-is-web-3-0 [pristupano: 14.04.2021.]

[10] „Introduction to the Semantic Web“ [Na Internetu]. *Ontotext.* Dostupno: https://ontotext.com/documents/SemTech-intro.pdf [pristupano 24.04.2021.]

[11] P. Hitzler, M. Krotzsch, S. Rudolph, *Foundations of Semantic Web Technologies.* Philadelphia, PA: Chapman & Hall / CRC, 2011

[12] J. Domingue, D. Fensel, J. A. Hendler, *Handbook of Semantic Web Technologies*. Berlin, Germany: Springer, 2011.

[13] B. Vuleta, „How Much Data is Created Every Day? [27 Staggering Stats]“. (28.01.2021.) [Na Internetu] *SeedScientific*. Dostupno: https://seedscientific.com/how-much-data-is-created-every-day/ [pristupano: 15.05.2021.]

[14] „Semantic Web Layer Cake“. [Slika] Dostupno: https://www.w3.org/2007/03/layerCake.png [pristupano: 19.05.2021.]

[15] G. Pandey, „The Semantic Web: An Introduction and Issues“, *International Journal of Engineering Research and Applications.* Izd. 1, str. 780 – 786, 2012. [Na Internetu]. Dostupno: https://www.academia.edu/1959581/IJERA_www_ijera_com_ [pristupano: 24.05.2021.]

[16] A. Maedche, S. Staab, „Ontology learning for the Semantic Web“, *IEEE Intelligent Systems,* izd. 16, br. 2, str. 72 – 79, 2001. [Na Internetu]. Dostupno: IEEE Xplore, https://ieeexplore.ieee.org/ [pristupano: 24.05.2021.]

[17] „Semantic Web Stack“. *Wikipedia.* [Slika] Dostupno: https://en.wikipedia.org/wiki/Semantic_Web_Stack#/media/File:Semantic_web_stack.svg [pristupano: 27.05.2021.]

[18] „Resource Description Framework (RDF): Concepts and Abstract Syntax“. (10.02.2004.) *World Wide Web Consortium.* [Na Internetu]. Dostupno: https://www.w3.org/TR/rdf-concepts/ [pristupano: 12.06.2021.]

[19] S. Powers, *Practical RDF*. Sebastopol, CA: O'Reilly, 2003.

[20] „RDF 1.1 Concepts and Abstract Syntax“ (25.02.2014.) *World Wide Web Consortium.* [Na Internetu]. Dostupno: https://www.w3.org/TR/rdf11-concepts/#section-Datatypes [pristupano: 20.06.2021.]

[21] „XML – Schemas“. *TutorialsPoint.* [Na Internetu]. Dostupno: https://www.tutorialspoint.com/xml/xml_schemas.htm [pristupano: 20.06.2021.]

[22] „RDF Schema 1.1“ (25.02.2014.) *World Wide Web Consortium.* [Na Internetu]. Dostupno: https://www.w3.org/TR/rdf-schema/ [pristupano: 01.07.2021.]

[23] B. McBride, „The Resource Description framework (RDF) and its vocabulary description language RDFS“. *Handbook on Ontologies*, Berlin: Springer Berlin Heidelberg, 2004., str. 51-65. [Na Internetu]. Dostupno: Springer Link, https://link.springer.com/ [pristupano: 01.07.2021.]

[24] L. W. Lacy, *Owl: Representing information using the web ontology language*. Victoria, BC, Canada: Trafford Publishing, 2005.

[25] G. Antoniou, F. van Harmelen, "Web ontology language: OWL". *Handbook on Ontologies*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, str. 67–92. [Na Internetu]. Dostupno: Springer Link, https://link.springer.com/ [pristupano: 01.07.2021.]

[26] T. Gruber, „Ontology". [Na Internetu]. Dostupno: https://tomgruber.org/writing/ontology-in-encyclopedia-of-dbs.pdf [pristupano: 05.07.2021.]

[27] B. DuCharme, *Learning Sparql*. O'Reilly Media, 2014. [Na Internetu]. Dostupno: https://www.oreilly.com/data/free/learning-sparql-2ed-sample-marklogic.pdf [pristupano: 07.07.2021.]

[28] "SPARQL Query Language for RDF", *W3C* [Na Internetu]. Dostupno: https://www.w3.org/TR/rdf-sparql-query/ [pristupano: 07.07.2021.]

[29] A. Bernstein, J. Hendler, and N. Noy, "A new look at the semantic web," Commun. ACM, vol. 59, no. 9, pp. 35–37, 2016. [Na Internetu] Dostupno: https://dl.acm.org/doi/fullHtml/10.1145/2890489 [pristupano: 27.07.2021.]

[30] M. Bramer, *Logic Programming with Prolog*, Springer, 2005. [Na Internetu] Dostupno: http://yuliana.lecturer.pens.ac.id/Kecerdasan%20Buatan/ebook/Logic%20Programming%20with%20Prolog.pdf [pristupano: 28.07.2021.]

[31] I. Bratko, Prolog Programming for Artificial Intelligence, 4th ed. Boston, MA: Addison-Wesley Educational, 2011. [Na Internetu]. Dostupno: https://silp.iiita.ac.in/wp-content/uploads/PROLOG.pdf [pristupano: 29.07.2021.]

[32] M. Loiseleur and N. Vigier, "Prolog," *Boklm.eu*. [Na Internetu]. Dostupno: https://boklm.eu/prolog/page_6 [pristupano: 31.07.2021.]

[33] S. Lampa, "SWI-Prolog as a Semantic Web Tool for semantic querying in Bioclipse: Integration and performance benchmarking," *Webb.uu.se*. [Na Internetu]. Dostupno: http://files.webb.uu.se/uploader/ibg.uu.se/74097_Lampa_Samuel_report.pdf. [pristupano: 01.08.2021.]

[34] "SWI-Prolog for the (semantic) web," Swi-prolog.org. [Na Internetu]. Dostupno: https://eu.swi-prolog.org/web/index.html. [pristupano: 01.08.2021.].

[35] J. Wielemaker, W. Beek, M. Hildebrand, J. van Ossenbruggen, "ClioPatria: A SWI-Prolog infrastructure for the Semantic Web," Semant. Web, izd. 7, br. 5, str. 529–541, 2016.

[36] Swi-prolog.org. [Na Internetu]. Dostupno: https://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/http.html%27). [pristupano: 30.08.2021.]

[37] T. Lager, J. Wielemaker, “Pengines: Web logic programming made easy,” Theory Pract. Log. Program., izd. 14, br. 4–5, str. 539–552, 2014.

[38] J. Wielemaker, “SWI - Prolog Semantic Web Library 3.0,” Swi-prolog.org. [Na Internetu]. Dostupno: https://www.swi-prolog.org/pldoc/doc_for?object=section(%27packages/semweb.html%27). [pristupano: 31.08.2021.]

[39] M. A. Musen i Protégé Team, “The protégé project: A look back and a look forward: A look back and a look forward,” AI Matters, izd. 1, br. 4, str. 4–12, 2015.

[40] “library(persistency): Provide persistent dynamic predicates,” Swi-prolog.org. [Na Internetu]. Dostupno: https://www.swi-prolog.org/pldoc/man?section=persistency. [pristupano: 01.09.2021.]

[41] D. Wynings, “RIP: The Semantic Web,” Diffbot.com (02.10.2017.) [Na Internetu]. Dostupno: https://blog.diffbot.com/rip-the-semantic-web/. [pristupano: 01.09.2021.]

[42] M. Sabou, “An introduction to semantic web technologies,” Semantic Web Technologies for Intelligent Engineering Applications, Cham: Springer International Publishing, 2016, str. 53–81.

[43] “OWL 2 web ontology language structural specification and functional-style syntax (second edition),” Www.w3.org. [Na Internetu]. Dostupno: https://www.w3.org/TR/owl2-syntax/. [pristupano: 08.09.2021.]

[44] “InverseFunctionalProperty - W3C Wiki,” Www.w3.org. [Na Internetu]. Dostupno: https://www.w3.org/wiki/InverseFunctionalProperty. [pristupano: 08.09.2021.]

[45] X. Wang, Z. M. Ma, F. Zhang, and L. Yan, “RIF centered rule interchange in the semantic web,” Lecture Notes in Computer Science, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, str. 478–486.

[46] “Trying to understand library(persistency),” SWI-Prolog forum, (15.04.2019.) [Na Internetu]. Dostupno: https://swi-prolog.discourse.group/t/trying-to-understand-library-persistency/542. [pristupano: 11.09.2021.]

[47] “Pengines,” Swi-prolog.org. [Na Internetu]. Dostupno: https://pengines.swi-prolog.org/docs/index.html. [pristupano: 11.09.2021.]

Popis slika