Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matic Repše

Pogovorni agent v slovenskem jeziku za sistem za upravljanje s človeškimi viri

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Zoran Bosnić

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidat naj v diplomi implementira pogovornega agenta, ki bo posrednik med uporabnikom in kadrovskim informacijskim sistemom. Agent naj deluje v slovenskem jeziku in naj se uporablja za povpraševanje po osnovnih kadrovskih podatkih, kot so npr. podatki o dopustu ipd. Pri implementaciji naj uporabi slovenska jezikovna orodja. Rešitev naj evalvira.

Iskrena zahvala gre izr. prof. dr. Zoranu Bosniću za njegov čas, odzivnost in strokovne nasvete pri nastajanju diplomskega dela. Zahvaljujem se tudi kolegom na CJVT, teti Moniki in punci Ivoni za pomoč pri skladenjskem označevanju stavkov. Hvala še sodelavcem za vso svetovanje in spodbudo ter družini in prijateljem, ki so mi tekom študija stali ob strani.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
NLP	natural language processing	obdelava naravnega jezika
HRM	human resources management	upravljanje s človeškimi viri
NLTK	natural language toolkit	knjižnica za obdelavo naravnega jezika
CLARIN.SI	common language resources and technology infrastructure, Slovenia	slovenska raziskovalna infrastruktura za jezikovne vire in tehnologije
XML	extensible markup language	razširljiv označevalni jezik
TEI	text encoding initiative	pobuda za kodiranje besedila
RDR	ripple down rules	metoda za učenje pravil za lematizacijo
MSTParser	minimum-spanning tree parser	razčlenjevalnik z minimalnim vpetim drevesom
JOS	linguistic labeling of slovene	jezikoslovno označevanje slovenščine
SOQL	salesforce object query language	jezik za povpraševanje salesforce objektov
SQL	structured query language	strukturirani povpraševalni jezik
HTTP	hypertext transfer protocol	protokol za prenos hiperteksta
URL	uniform resource locator	enolični krajevnik vira
HTTPS	HTTP Secure	varni HTTP
REST	representational state transfer	arhitektura za izmenjavo podatkov med spletnimi storitvami
JSON	javascript object notation	oblika za izmenjavo podatkov
DOM	document object model	objektni model dokumenta
CRM	customer resources management	upravljanje odnosov s strankami

Povzetek

Naslov: Pogovorni agent v slovenskem jeziku za sistem za upravljanje s človeškimi viri

Povzetek: V diplomski nalogi smo se lotili implementacije pogovornega robota v slovenščini, ki služi kot vmesnik za hitro iskanje informacij v zaprtem kadrovskem sistemu. Za implementacijo agenta smo uporabili razčlenjevalnik, razvit pri projektu Sporazumevanje v slovenskem jeziku (2008-2013). Razčlenjen vnosni stavek smo obdelali v programskem jeziku Python in mu poiskali vse stavčne člene. Na podlagi najdenih stavčnih členov smo nato poizkusili razumeti, po katerem podatku sprašuje vnosni stavek. V primerih, kjer smo uspešno našli pomen, smo iskani podatek poiskali v kadrovskem sistemu ter uporabniku vrnili odgovor. Preko razvitega vmesnika lahko uporabniki kadrovskega sistema sprašujejo po podatkih, ki so jim v sklopu sistema na voljo z nekaj kliki. Pri evalvaciji smo si pomagali z ročno skladenjsko in oblikoslovno označenim učnim korpusom, ki smo ga ustvarili v sklopu razvoja. Rezultati so pokazali, da smo iskanje pomena zastavili dobro. Z združitvijo našega učnega korpusa in učnega korpusa ssj500k smo uspešnost iskanja pomena dvignili na 84 %.

Ključne besede: obdelava naravnega jezika, pogovorni agent, HRM.

Abstract

Title: Slovene chat agent for a human resources management system

In this thesis we attempted to implement a slovene chat agent. The agent would serve as an interface to quickly retrieve data from a closed HRM (human resources management) system. To implement the mentioned agent, we used a slovene parser developed in the scope of the Communication in Slovene project. The parsed input sentence was then processed in Python, where we found all of its sentence elements. Knowing its sentence elements, we then tried to understand, which data it was asking for. In cases where we successfully found the meaning, we searched for the wanted data in the HRM system and then answered back to the user if we found it. Through the developed interface, users of the closed HRM system can ask for information with natural slovene language. We tested the developed agent with the new learning corpus we created during the development. Results showed we set up a good meaning searching algorithm. With the merging of our new learning corpus and the learning corpus ssj500k we raised the success rate of our meaning searching algorithm up to 84 %.

Keywords: NLP, chat agent, HRM.

Poglavje 1

Uvod

Živimo v času množične digitalizacije številnih procesov, ki nas obkrožajo (npr. procesi v zdravstvu, procesi pri prodaji in odnosi s strankami (CRM), procesi pri upravljanju s človeškimi viri (HRM) itd.). Tam, kjer je do digitalizacije že prišlo, se vedno bolj stremi k optimizaciji ter pospešitvi teh procesov. To pomeni, da morajo implementatorji teh sistemov skrbeti za hitrost aplikacij in ustreznost uporabniških vmesnikov. Tako je pri podjetju, ki strankam nudi sistem HRM, prišlo do potrebe, da bi uporabniki lahko na hitrejši in enostavnejši način pridobivali podatke, ki so jim v sistemu sedaj na voljo s pomočjo nekaj klikov po aplikaciji.

Ker so grafični uporabniški vmesniki že sedaj hitri in enostavni za uporabo, je naslednji korak pohitritve procesa pridobivanja podatkov spraševanje sistema po podatkih z naravnim jezikom in odgovarjanje na vprašanja z inteligentnim pogovornim agentom (robotom). Naravni jezik je vsak jezik, ki je nastal naravno z uporabo govora in nato pisanja tekom razvoja človeštva.

Uporabnik bi v nekakšen enostavno dostopen vmesnik vpisal svoje vprašanje (npr. "Koliko imam letos še dopusta?"), na katero bi potem hitro dobil odgovor. Za ustrezen odgovor bo v ozadju vmesnika tekla aplikacija z dostopom do sistema HRM. Preden pa bo iskala podatek v omenjenem sistemu, bo morala razčleniti in razumeti uporabnikov vnosni niz. Za ta del bomo uporabili obdelavo naravnega jezika. Ko bo sistem razumel, po

katerem podatku sprašuje uporabnik, bo le-tega vrnil.

Ker je omenjeno podjetje slovensko, in ker so večina njegovih strank slovenska podjetja, je smiselno, da se aplikacija implementira v slovenskem jeziku. To predstavlja poseben izziv, saj ne obstaja noben odprtokodni primer implementacije inteligentnega pogovornega agenta (robota) v slovenščini.

1.1 Opis problema

Glavni izziv naše ciljne implementacije bo uspešno razumevanje uporabniških poizvedb, podanih v slovenskem jeziku. Računalnik sam po sebi ne more razumeti naravnega jezika. Dolgo časa je to bila tudi precej velika ovira. Ob stvaritvi modernejših algoritmov umetne inteligence (strojno učenje, globoko učenje, podatkovno rudarjenje itd.) je ta ovira postala lažje premostljiva. Pojavlja se vse več odprtokodnih knjižnic, ki vsebujejo implementacijo teh algoritmov in z njimi rešujejo probleme obdelave in razumevanja naravnega jezika [20].

Kmalu zatem so se razvile tudi knjižnice za pogovorne robote [3], saj temeljijo na prej omenjenih knjižnicah. V to množico spadajo tudi inteligentni pogovorni agenti (roboti).

Tukaj je problematično to, da so ustvarjene knjižnice po večini razvite za obdelavo angleščine in ne slovenščine. Ena izmed redkih knjižnic, ki delno podpirajo obdelavo slovenščine, je knjižnica NLTK (Natural language toolkit) [22] v Pythonu. A tudi ta nam ne bo kaj dosti v pomoč, saj na primer ni sposobna lematizacije (iskanja osnovne oblike besede) slovenskih besed [20]. Ta pa je ena izmed predpogojev za dobro obdelavo naravnega jezika.

Potrebujemo torej učinkovito orodje za obdelavo slovenščine. Slovenščina je eden od redkih jezikov, ki je vseskozi ohranil dvojino kot slovnično število, pa naj bo to za sklanjanje pridevnikov, samostalnikov, številkov in zaimkov ali pa za spreganje glagolov. Posledice te prepoznavne značilnosti se kažejo tudi v računalniški obdelavi naravnega jezika. Slovenščina spada tudi med bolj pregibne jezike, saj pozna šest sklonov in tri spole. To pomeni, da imajo

samostalniki in pridevniki veliko več možnih oblikoskladenjskih oznak kot v angleščini, zato je tudi zahtevnost obdelave slovenščine večja [19].

Problem obdelave naravnega jezika v slovenščini bomo zato poizkusili reševati z razčlenjevalnikom za slovenski jezik. Razčlenjevalnik je bil razvit v Sloveniji v sklopu projekta Sporazumevanje v slovenskem jeziku (2008-2013) [1]. Uporabili ga bomo za oblikoslovno (iskanje oblikoskladenjskih oznak besed) in skladenjsko (stavčni členi in razmerja besed znotraj posameznega stavčnega člena) označevanje ter za lematizacijo uporabniškega vnosnega stavka.

Ko imamo enkrat vnosni stavek razčlenjen, lahko s pomočjo najdenih stavčnih členov ter lem in oblikoskladenjskih oznak besed poizkusimo razumeti, po katerem podatku sprašuje. Ker gre za zaprt sistem, imamo končno število možnih iskanih podatkov. To pomeni, da lahko od uporabnikov pričakujemo po čem bodo povpraševali, in zato pri iskanju pomena iščemo samo določene ključne besede ter njihove sopomenke.

Poglavje 2

Pregled področja

2.1 Pogovorni roboti

V svetu najdemo mnogo implementacij inteligentnih pogovornih agentov (robotov). Trenutno so med najbolj znanimi Siri [15], ki ga ponuja Apple na vsaki novejši napravi, nekaj podobnega so implementirali pri Amazonu [14] in tudi pri Microsoftu [21]. Ti boti so precej kompleksnejši kot naša ciljna implementacija, saj so namenjeni splošni uporabi vsakemu, ki uporablja njihove naprave. Naš sistem pa bo moral odgovarjati le na vprašanja zaprtega tipa, o tem, kar je uporabniku dostopno znotraj sistema.

Če malo pobrskamo po spletu, hitro najdemo glavnega ponudnika takšnega sistema; to je podjetje Artificial Solutions. Ponujajo platformo Teneo [27], na podlagi katere potem strankam pomagajo zgraditi inteligentne pogovorne agente (robote), ki se lahko naučijo petintridesetih jezikov. Slednji so zmožni odgovarjati na kratka in jedrnata vprašanja uporabnikov o zaprti temi, ki jo izbere stranka.

Veliko podjetij v svetu se odloča za njihovo rešitev, med njimi so na primer tudi Bosch, Shell, IKEA itd. Prav tako pa med njimi najdemo tudi dve implementaciji v slovenščini, to sta davčna asistentka Vida [7] in TIA - Telekomova interaktivna asistentka [28]. Čeprav sta obe implementaciji platforme Teneo po specifikaciji zelo podobni naši ciljni aplikaciji, nam to

pove le, da je naš cilj izvedljiv in nič drugega, saj je rešitev s platformo Teneo zaprtokodna.

Podoben produkt, kot je platforma Teneo, so izdelali tudi pri našem domačem podjetju Amebis d. o. o., Kamnik. Imenuje se SecondEgo [2]. SecondEgo se lahko nauči kateregakoli jezika, ima pa tudi predznanje iz jezikov angleščine, slovenščine, nemščine in francoščine. Nauči se lahko odgovarjati na vprašanja zaprtega tipa, na temo, ki jo določi stranka. Primerov implementacije v slovenščini je kar nekaj: Klepec [2], Modri Miha (`www.modra-zavarovalnica.si`), Stane (`www.promet.si`) itn.

V nadaljevanju bomo predstavili nekaj zgoraj omenjenih primerov implementacije vprašalno-odgovornih robotov v slovenščini.

2.1.1 Davčna asistentka Vida

Prvi primer takšnega sistema se je pri nas pojavil leta 2007 s strani davčne uprave kot pomoč pri odgovarjanju na vprašanja uporabnikov na njihovi takratni spletni strani. To je bila davčna asistentka Vida [7], ki so jo razvili pri Artificial Solutions.

Odgovarjala je na jasno strukturirana vprašanja, ki so se po večini nanašala na tematiko v zvezi z davčno upravo (npr. "Kaj je informativni izračun?"). Za obdelavo uporabniškega vnosa in izdelavo odgovorov na uporabniški vnos naj bi uporabljala umetno inteligenco.

Asistentko so leta 2014 "upokojili", saj naj bi bila prevelik strošek za vzdrževanje (začetna implementacija je stala več kot 600.000 EUR, letno vzdrževanje pa več kot 80.000 EUR).

2.1.2 TIA - Telekomova Interaktivna Asistentka

Drugi takšen primer implementacije platforme Teneo je Telekomova TIA [28]. Projekt je zaživel leta 2009 in je zelo podoben davčni asistentki Vidi, saj tudi ta odgovarja na jasno strukturirana vprašanja. Razlika je v tematiki, na katero zna podati odgovor, to je samoumevno.

TIA odgovarja na vprašanja v zvezi s paketi, ki jih ponuja Telekom ali pa obiskovalca kar preusmeri na stran, za katero meni, da vsebuje odgovor. Tako npr. TIA ob vprašanju: "Kateri paket je najbolj ustrezen za študente?" uporabnika preusmeri na stran, ki vsebuje informacijo o tem, kakšne popuste lahko dobijo študenti.

2.1.3 Klepec

Klepec [2] je primer implementacije sistema SecondEgo. Ta ni bil implementiran za specifično stranko, ampak služi kot virtualni asistent na domači strani podjetja Amebis. Sprašujemo ga lahko o njihovih produktih.

Podobno kot Vida in TIA, tudi Klepec večinoma razume samo kratka in jedrnata vprašanja. Primer prikazuje slika 2.1, kjer smo Klepcu na dva različna načina postavili vprašanje o ceni izdelka SecondEgo, enkrat kratko in jedrnato in drugič ravno nasprotno. V prvem primeru je odgovoril pravilno, v drugem pa vprašanja ni razumel v celoti, vseeno pa je razumel, da sprašujemo po izdelku SecondEgo.

2.2 Analiza besedil

Na temo obdelave naravnega jezika v slovenščini najdemo vsaj štiri sorodna dela, s katerimi si lahko pomagamo pri implementaciji ciljnega sistema. Sorodna dela sledijo v nadaljevanju.

2.2.1 Knjižnice v Pythonu

V diplomski nalogi [20] so avtorji raziskovali, katera programska knjižnica v Pythonu je najbolj primerna za obdelavo naravnega jezika. Med seboj so primerjali sledeče knjižnice:

- NLTK,
- SpaCy,

Slika 2.1: Primer pogovora z virtualnim asistentom Klepcem na spletni strani podjetja Amebis d. o. o., Kamnik. Na levi strani smo Klepcu postavili kratko in jedrnato vprašanje, na katero je tudi pravilno odgovoril. Na desni strani smo mu postavili istopomensko vprašanje na malce drugačen, bolj zapleten način. Tega Klepec ni razumel.

- PyNLPl,
- Pattern,
- TextBlob.

Primerjali so jih na podlagih več kriterijev (tokenizacija, lematizacija, oblikoskladenjsko označevanje itd.). Med drugim so ugotavljali tudi, katera je najbolj primerna za obdelavo slovenskega jezika, pri čemer so odkrili, da je to knjižnica NLTK. Ta lahko oblikoslovno in skladenjsko označuje, vendar za to potrebuje učni korpus v slovenščini.

Čeprav je ta knjižnica med omenjenimi najbolj primerna za obdelavo slovenskega jezika, je za nas neprimerna, saj ni zmožna lematizacije slovenskih besed. Lematizacija je za nas pomembna predvsem zaradi zmanjšanja zahtevnosti pri iskanju pomena posamezne besede.

2.2.2 Slovenski jezik v digitalni dobi

Delo [19] govori o pomembnosti jezikovnih tehnologij v Evropi, govori tudi o zastopanosti slovenščine na internetu in o tem, kakšne nevarnosti pretijo jezikom, ki so na spletu slabše zastopani. Razloži tudi, zakaj je obdelava naravnega jezika za programerje zahtevna operacija ter zakaj je obdelava slovenščine bolj zahtevna od naravne angleščine. Največ za nas pomembnih podatkov pa izvemo v poglavju o jezikovnih tehnologijah za slovenščino.

V omenjenem poglavju knjiga prepozna najbolj pomembne jezikovne tehnologije, ki so:

- preverjanje črkovanja,
- podpora sestavljanju besedil,
- računalniško podprto učenje jezikov,
- informacijsko poizvedovanje,
- luščenje informacij,

- avtomatsko povezemanje,
- avtomatsko odgovarjanje na vprašanja,
- prepoznava govora,
- sinteza govora.

Njihovo arhitekturo poenoti in poenostavi na štiri module, ki so:

1. predobdelava,
2. slovnična analiza,
3. semantična analiza in
4. modul, ki je za vsako jezikovno tehnologijo drugačen.

Za problem avtomatskega odgovarjanja na vprašanja, ki ga večinoma lahko poenotimo z našim, četrti zgornji modul predstavlja iskanje pomena uporabnikovega vnosa in nato tudi iskanje odgovora.

Avtomatsko odgovarjanje na vprašanja v knjigi predstavlja precej širši pomen od našega, saj je obravnavan kot modul pri iskanju po spletu, ta pa je praktično neskončen v primerjavi z našim zaprtim kadrovskim sistemom. Ko v našem primeru najdemo pomen vprašanega stavka, lahko takoj vemo, ali imamo ta podatek ali ne. V primeru iskanja odgovora na spletu pa tukaj vskočijo še druge pomembne veje jezikovnih tehnologij, te so: luščenje podatkov, avtomatsko povzemanje ter tvorba besedila.

Avtor nam v poglavju predstavi tudi, kako slabo razvite so bile zgoraj naštete jezikovne tehnologije pri nas v letu izdaje (2012) v primerjavi z drugimi jeziki v Evropi (npr. nemščina, francoščina itd.). Do danes so se stvari obrnile na bolje. V letu 2014 je nastal slovenski konzorcij CLARIN.SI kot član evropskega konzorcija CLARIN [4]. Na njihovi spletni strani so na voljo vsi jezikovni viri in tehnologije v slovenščini, ki so bili do sedaj razviti s strani članov konzorcija.

2.2.3 Statistična analiza slovenskih jezikovnih korpusov

Glavna motivacija diplomske naloge [18] je bila učinkovita in hitra statistična obdelava korpusov, ki vsebujejo od nekaj sto do več miljard besed. Razvili so orodje, ki je lahko nad slovenskimi velikimi jezikovnimi korpusi izvajalo statistično analizo. Pomagali so si z vsemi razpoložljivimi kapacitetami, ki jih je pogonski sistem imel na voljo (paralelizacija, izraba pomnilnika).

Prebrane stavke in besede iz korpusa so v Javi programsko predstavili z objektoma *stavek* in *beseda*. Objekt *stavek* je vseboval atribut *besede*, ta predstavlja seznam vseh objektov *besed*, ki se pojavijo v stavku. Objekt *beseda* pa vsebuje atribute: *beseda* (znakovni niz besede), *lema* (znakovni niz osnovne oblike besede) in *msd* (tabela znakov, ki predstavljajo oblikoskladenjsko oznako besede). Vsi ti podatki so v korpusu v obliki XML-TEI [5], več o tem v podpoglavju o jezikovnih orodjih.

Podobno programsko predstavitev vnosnih stavkov in besed bomo poizkusili implementirati v našem programu, v jeziku Python.

2.2.4 Abstraktivno povzemanje dokumentov v slovenskem jeziku

Diplomska naloga [16] govori o avtomatizaciji postopka povzemanja dokumentov v slovenskem jeziku. To vejo jezikovne tehnologije smo zgoraj že omenjali. Motivi za ciljno implementacijo so bili: količina elektronskih dokumentov, ki jih imamo na voljo na spletu ter želja po hitri izluščitvi bistva omenjenih dokumentov. Tu gre predvsem za pridobivanje časa v primerjavi z dosedanjim ročnim povzemanjem dokumentov.

Za obdelavo naravnega jezika so uporabili razčlenjevalnik za slovenski jezik [1]. Razčlenjenim stavkom so nato poiskali njihove povedke, osebke in predmete. To nam je v pomoč, saj bo tudi pri nas po uporabi razčlenjevalnika potrebno razčlenjenim stavkom poiskati stavčne člene.

2.3 Jezikovna orodja

Pri razvoju ciljne aplikacije bomo uporabili jezikovne vire in tehnologije, ki so jih razvili v okviru obsežnega projekta Sporazumevanje v slovenskem jeziku (2008-2013) [1]. Projekt je izvajalo podjetje Amebis, d. o. o., Kamnik, financiran pa je bil s strani Evropske unije ter s strani Ministrstva za izobraževanje, znanost in šport. Podjetje Amebis smo zgoraj že omenjali kot ponudnika virtualnega asistenta SecondEgo [2].

Uporabili bomo skladenjski razčlenjevalnik za slovenski jezik [1] in učni korpus ssj500k [1]. Posredno bomo tako uporabili tudi oblikoslovni označevalnik Obeliks [1], ki je implementiran v razčlenjevalniku.

Omenjene jezikovne vire in tehnologije bomo opisali v nadaljevanju.

2.3.1 Oblikoslovni označevalnik Obeliks

Obeliks[1] je samostojen program, namenjen oblikoslovnemu označevanju besedila. Vhodno besedilo razdeli na stavke in besede, katerim nato pripiše oblikoskladenjsko oznako ter lemo. Rezultat je oblikoslovno označeno besedilo v formatu XML-TEI (slika 2.2).

Označevalnik deluje po principu nadzorovanega strojnega učenja. To pomeni, da se lahko nauči označevati s pomočjo ročno oblikoslovno že označenega učnega korpusa. Z njim v fazi učenja zgradi model znanja, ki pa se potem uporabi za uvrščanje neoznačenih vhodnih besedil.

Lematizacija besed je implementirana z orodjem LemmaGen, ki je nastalo pri diplomski nalogi: "Implementacija učinkovitega sistema za gradnjo, uporabo in evaluacijo lematizatorjev tipa RDR" [17].

2.3.2 Skladenjski razčlenjevalnik za slovenski jezik

Skladenjski razčlenjevalnik za slovenski jezik nam nudi osnovne operacije za učinkovito obdelavo naravnega jezika v slovenščini. Kot smo že ome-

[1]Več o Obeliksu in o tem, kako deluje, si lahko preberete na spletni strani projekta (`http://www.slovenscina.eu`).

```xml
<TEI
    xmlns="http://www.tei-c.org/ns/1.0">
    <text>
        <body>
            <p>
                <s>
                    <w msd="Rsn" lemma="koliko">Koliko</w>
                    <S/>
                    <w msd="Somer" lemma="dopust">dopusta</w>
                    <S/>
                    <w msd="Ggnspe-n" lemma="imeti">imam</w>
                    <S/>
                    <w msd="Rsn" lemma="letos">letos</w>
                    <c>?</c>
                </s>
            </p>
        </body>
    </text>
</TEI>
```

Slika 2.2: Rezultat oblikoslovnega označevanja Obeliksa, za vhodni stavek: "Koliko dopusta imam letos?".

nili, so te operacije: lematizacija, oblikoslovno označevanje ter skladenjsko označevanje.

Skladenjsko razčlenjevanje temelji na odprtokodnem razčlenjevalniku MST-Parser [10], ki išče minimalno vpeto drevo v usmerjenih grafih. Ta si s pomočjo ročno oblikoslovno označenega in skladenjsko razčlenjenega korpusa zgradi model znanja. Zgrajeni model znanja se nato uporabi pri razčlenjevanju oblikoslovno označenih besedil.

Za skladenjsko razčlenjevanje je uporabljen sistem odvisnostnih drevesnic, razvit pri projektu JOS (Jezikoslovno označevanje slovenščine) [8]. V njem so skladenjski odnosi za posamezno poved predstavljeni z drevesno strukturo, vsaka povezava v drevesu pa pripada enemu izmed desetih tipov povezav, ki so predstavljene v tabeli 2.1.

Vnos programa je lahko tekstovna datoteka ali oblikoslovno označeno besedilo v formatu XML-TEI (slika 2.2). Za skladenjsko označevanje program potrebuje besedilo v omenjenem formatu. V primeru, da je vnos tekstovna datoteka, se za predobdelavo uporabi Obeliks, saj vrne ustrezen format. V nasprotnem primeru pa se ta del izpusti. Rezultat razčlenjevanja besedila je

oblikoslovno označeno in skladenjsko razčlenjeno besedilo v formatu XML-TEI (slika 2.3).

Skupina povezav	**Tip povezave**	**Kaj povezuje**
Povezave prvega nivoja označujejo razmerja znotraj besednih zvez.	dol	Jedro in določilo besednih zvez.
	del	Deli zloženega povedka.
	prir	Jedra v prirednih zvezah znotraj stavka.
	vez	Besede ali ločila v vezniški vlogi.
	skup	Nepolnopomenske besede, ki imajo zelo močno tendenco po sopojavljanju.
Povezave drugega nivoja označujejo stavčne člene.	ena	Osebek stavka.
	dve	Predmet stavka.
	tri	Prislovno določilo lastnosti.
	štiri	Ostala prislovna določila.
Povezava tretjega nivoja se uporablja za povezovanje vseh ostalih struktur.	modra	Hierarhično najvišje pojavnice, skladenjsko manj predvidljive in oddaljene strukture, vrinki, ločila.

Tabela 2.1: Tipi povezavi pri skladenjskem razčlenjevanju.

2.3.3 Učni korpus ssj500k

Učni korpusi se uporabljajo pri grajenju modelov znanja za programe, ki izvajajo strojno analizo besedil. Naučeni model znanja se nato uporabi za analizo novih neoznačenih besedil. Dva primera programa, ki na takšen način uporabljata učni korpus, smo že omenjali, to sta Obeliks in Skladenjski razčlenjevalnik za slovenski jezik. Vsebina učnih korpusov so strukturirane zbirke besedil, v katerih so ročno pregledani podatki jezikoslovne narave. Korpusi so v formatu XML-TEI [5], ki je namenjen standardizaciji določanja strukture korpusom.

Učni korpus ssj500k [1] vsebuje celotni korpus jos100k in 400.000 besed iz korpusa jos1M, oba sta nastala pri projektu JOS [8]. Vsako besedilo v ssj500k je razdeljeno na odstavke, stavke in besede, pri čemer sta vsaki besedi pripisani še lema in oblikoskladenjska oznaka. Ssj500k uporablja tabelo

```xml
<TEI
    xmlns="http://www.tei-c.org/ns/1.0">
    <text>
        <body>
            <p xml:id="0">
                <s xml:id="0.0">
                    <w lemma="koliko" msd="Rsn" xml:id="0.0.1">Koliko</w>
                    <S />
                    <w lemma="dopust" msd="Somer" xml:id="0.0.2">dopusta</w>
                    <S />
                    <w lemma="imeti" msd="Ggnspe-n" xml:id="0.0.3">imam</w>
                    <S />
                    <w lemma="letos" msd="Rsn" xml:id="0.0.4">letos</w>
                    <c xml:id="0.0.5">?</c>
                    <links>
                        <link afun="dol" dep="0.0.1" from="0.0.2" />
                        <link afun="štiri" dep="0.0.2" from="0.0.3" />
                        <link afun="modra" dep="0.0.3" from="0.0.0" />
                        <link afun="štiri" dep="0.0.4" from="0.0.3" />
                        <link afun="modra" dep="0.0.5" from="0.0.0" />
                    </links>
                </s>
            </p>
        </body>
    </text>
</TEI>

```

Slika 2.3: Rezultat skladenjskega razčlenjevalnika za slovenščino pri vhodnem stavku: “Koliko dopusta imam letos?”.

oblikoskladenjskih oznak, ki so jih določili pri projektu JOS. Vseh možnih oblikosladenjskih oznak je nekaj več kot 1900. V primerjavi z angleščino, kjer jih je nekaj več kot 70, je to ogromno. Zaradi tega je tudi obdelava slovenskih besedil precej zahtevnejša od angleških.

V ssj500k je tudi 11.411 skladenjsko razčlenjenih stavkov in podatki, namenjeni učenju razpoznavanja imenskih entitet. Vsi ti podatki so ročno pregledani.

Ssj500k je bil uporabljen pri izgradnji modela znanja za program Obeliks. Del korpusa, ki je skladenjsko razčlenjen, pa je bil uporabljen pri izgradnji modela znanja za skladenjski razčlenjevalnik za slovenski jezik. Ta del korpusa je na voljo tudi v sklopu programske opreme razčlenjevalnika.

2.4 Opis ostalih tehnologij

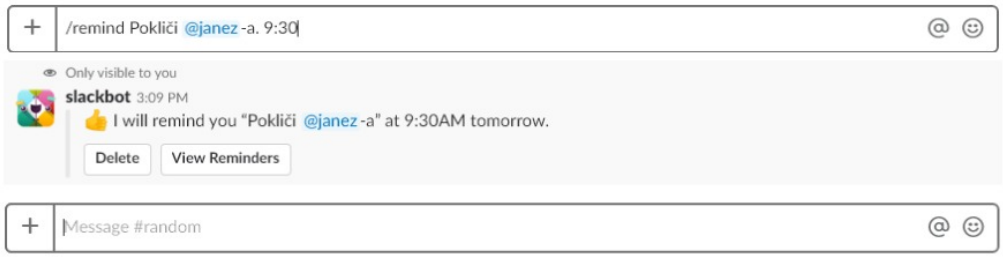
Obdelavo uporabniških vnosov bomo v kombinaciji z omenjenimi jezikovnimi orodji implementirali v programskem jeziku Python. Poleg tega potrebujemo vmesnik, preko katerega bomo uporabnikom na enostaven način omogočili vnos poizvedb z uporabo naravnega jezika; ta vmesnik bo Slack. Tukaj je potrebno omeniti tudi platformo Force.com, na kateri je razvit kadrovski sistem, in tehnologije, ki jih ta ponuja.

V nadaljevanju bomo opisali omenjene tehnologije in okolja.

2.4.1 Slack

Slack [26] je oblačna aplikacija, ki je namenjena enostavni interni komunikaciji med zaposlenimi v podjetjih. Število podjetij, ki jo uporabljajo, narašča, med njimi pa je tudi podjetje, ki ponuja omenjen kadrovski sistem ter veliko njihovih strank.

Pri Slacku so za svoje uporabnike razvili dober aplikacijski programski vmesnik, ki omogoča hitre integracije spletnih servisov in razvoj lastnih aplikacij za uporabo znotraj Slacka. Ena izmed možnih funkcionalnosti, ki jih ponujajo, so tudi ukazi s poševnico ("slash commands") - te bomo zaradi enostavnosti implementacije uporabili mi. Enega izmed že vgrajenih ukazov s poševnico prikazuje slika 2.4.



Slika 2.4: Slika prikazuje vgrajeni ukaz s poševnico, ki uporabniku omogoča izdelavo opomnikov. Na zgornjem delu slike je prikazan primer ukaza, ki ga uporabnik vpiše v polje za sporočila, v spodnjem delu pa rezultat izvedenega ukaza.

2.4.2 Platforma Force.com

Force.com [24] je oblačna platforma, ki jo je razvilo podjetje Salesforce.com, Inc. Na njej je mogoče zelo hitro razvijati podjetne rešitve, saj razvijalcem ni potrebno skrbeti za infrastrukturo in se tako lahko osredotočijo na vsebino.

Razvijalcem je v sklopu platforme na voljo več različnih ospredjih in zalednih tehnologij in aplikacijskih programskih vmesnikov (npr. Visualforce (ospredje), Lightning komponente (ospredje), Apex (zaledje) itn.) [25], s katerimi lahko razvijajo aplikacije. Nas najbolj zanima zaledni programski jezik Apex, saj bomo z njim naši aplikaciji omogočili dostop do kadrovskega sistema in do podatkov, po katerih poizvedujejo uporabniki.

Apex je objektni programski jezik, ki je po svoji naravi zelo podoben Javi. Vsak objekt, ki ga ustvarimo v podatkovnem modelu, takoj dobi tudi svoj objekt in konstruktorje za izdelavo, posodabljanje ter izbris. S tem postane razvijanje zaledne logike enostavno. Prav tako je v Apexu mogoče neposredno nad podatkovno bazo izvajati poizvedbe po podatkih s poizvedbenim jezikom SOQL (Salesforce object query language), ki pa je zelo podoben jeziku SQL. Spodaj je prikazan primer poizvedbe SOQL v Apexu, ki med zaposlenimi išče vse z imenom Janez.

```
List<Zaposleni> vsiJanezi = [SELECT Id, Ime, Priimek
                             FROM Zaposleni
                             WHERE Ime="Janez"];
```

Poglavje 3

Implementacija inteligentnega pogovornega agenta v slovenščini

Funkcionalnost našega sistema bo uporabnikom na voljo skozi uporabniški vmesnik aplikacije Slack. Ta je tukaj smiselna izbira predvsem zato, ker ga za interno komunikacijo med zaposlenimi uporablja veliko strank omenjenega podjetja in so tako njihovi zaposleni navajeni uporabe aplikacije. Prav tako nam Slack nudi dobro podporo za hiter razvoj ukazov s poševnico, s katerimi bomo zajeli uporabniško poizvedbo in vrnili odgovor aplikaciji. Za obdelavo uporabniškega vnosa bomo uporabili razčlenjevalnik za slovenski jezik ter programski jezik Python. V primeru, da bo obdelava vnosa ugotovila, po katerem podatku poizveduje uporabnik, bomo podatek pridobili iz kadrovskega sistema s programskim jezikom Apex. V nasprotnem primeru pa bomo uporabnika o neuspehu poizvedbe primerno obvestili.

3.1 Načrt razvoja aplikacije

Našo aplikacijo smo modularno razdelili na štiri dele:

1. arhitekturno ogrodje aplikacije,

2. razčlenjevanje uporabniških poizvedb,

3. algoritem za razumevanje razčlenjenih poizvedb in

4. pridobivanje podatkov iz kadrovskega sistema.

V nadaljevanju bomo opisali načrtovanje razvoja posameznega modula.

3.1.1 Arhitekturno ogrodje aplikacije

V Slacku bomo s pomočjo ukaza s poševnico zajeli uporabniško poizvedbo, podano z uporabo naravnega jezika. Ukaz s poševnico bo nato zajeti vnos posredoval našemu programu in čakal na odgovor. Tega je potrebno vrniti v treh sekundah. Če tega ne storimo, bo Slack uporabniku javil napako. Ker naš program verjetno ne bo mogel zagotoviti tako hitre obdelave vnosa in poizvedbe podatkov, bo tej povezavi odgovoril s HTTP statusom 200 ter informativnim stavkom za prikaz uporabniku (npr. “Obdelujem poizvedbo.”). Ukaz s poševnico nam poleg uporabniške poizvedbe posreduje tudi povratni URL, na katerega lahko v roku 30 minut posredujemo končni rezultat obdelave poizvedbe. Ko bo naš program razumel poizvedbo, jo bo skupaj s povratnim URL-jem posredoval v kadrovski sistem. Tam bomo v Apexu pridobili podatke in jih vrnili nazaj v Slack preko povratnega URL-ja.

3.1.2 Razčlenjevanje uporabniških poizvedb

Za obdelavo poizvedbe v slovenščini bomo uporabili razčlenjevalnik za slovenski jezik. Ker je razčlenjevalnik implementiran v Javi in ker vhodno besedilo za razčlenitev sprejema v obliki tekstovne datoteke, bo tukaj treba implementirati ovoj v Pythonu. Ta bo razčlenjevalnik ustrezno pognal in na koncu pridobil razčlenjeno besedilo za možnost nadaljnje obdelave.

3.1.3 Algoritem za razumevanje razčlenjenih poizvedb

Kot smo že omenjali, imamo opravka z zaprtim kadrovskim sistemom, ki ima posledično tudi končno število možnih podatkov za poizvedbe uporab-

nikov. Zato smo se tukaj odločili, da bomo za prepoznavanje iskanega podatka uporabili ročno napisana pravila. Druga možnost bi tukaj bila strojno učenje ali kombinacija obeh. Preden bomo začeli s prepoznavanjem iskanega podatka, bomo iz nabora možnih tem v kadrovskem sistemu izbrali eno, na katero bo naša rešitev pretežno osredotočena na začetku. Ko bo program okoli izbrane teme zadovoljivo prepoznaval pomen poizvedb, se bodo postopoma dodale še ostale. Znotraj vsake dodane teme bo treba določiti tudi podatke, po katerih bodo lahko spraševali uporabniki. Ko bomo imeli določeno začetno temo in začetni sklop podatkov, bomo potrebovali testna vprašanja, pridobili jih bomo s pomočjo znancev. S testnimi vprašanji bomo poizkusili prepoznati razne trende in lastnosti, ki se v njih pojavljajo. Z ugotovljenim bomo tako lažje pisali pravila za prepoznavanje iskanega podatka. Prav tako bomo vprašanja potrebovali za testiranje našega programa. Algoritem želimo razvijati s pomočjo pravilno skladenjsko in oblikoslovno označenih primerov. Ker ne vemo, kako natančno trenutni model znanja razčlenjuje vprašalne stavke, bomo razčlenjevalnik za čas razvoja obšli ter iz testnih vprašanj izdelali ročno pregledan oblikoslovno označen in skladenjsko razčlenjen učni korpus. Ta nam bo tekom razvoja služil kot nadomestilo avtomatsko razčlenjenih stavkov, v pomoč pa nam bo tudi kasneje pri testiranju natančnosti razčlenjevanja. Ko bomo imeli vse našteto pripravljeno, se bomo lotili razvoja algoritma za razumevanje razčlenjenih poizvedb.

Razčlenjeno poizvedbo bomo pretvorili v programske objekte *Stavek*, *Beseda* in *StavcniClen*, saj bo z njimi obdelava poizvedbe lažja. Do te mere narejen program bomo pognali z vhodom celotnega novega učnega korpusa. Tako bomo na hitro dobili vse stavčne člene testnih vprašanj. To nam bo v pomoč, ker bomo videli, kateri stavčni členi so najbolj pogosti in katere lahko v sklopu te teme pričakujemo. Tu bo potrebno prepoznati in napisati pravila za prepoznavanje pomena. Ob kasnejšem dodajanju tem bo postopek do tu treba ponoviti ter morebitna manjkajoča pravila dopisati v program.

3.1.4 Pridobivanje podatkov iz kadrovskega sistema

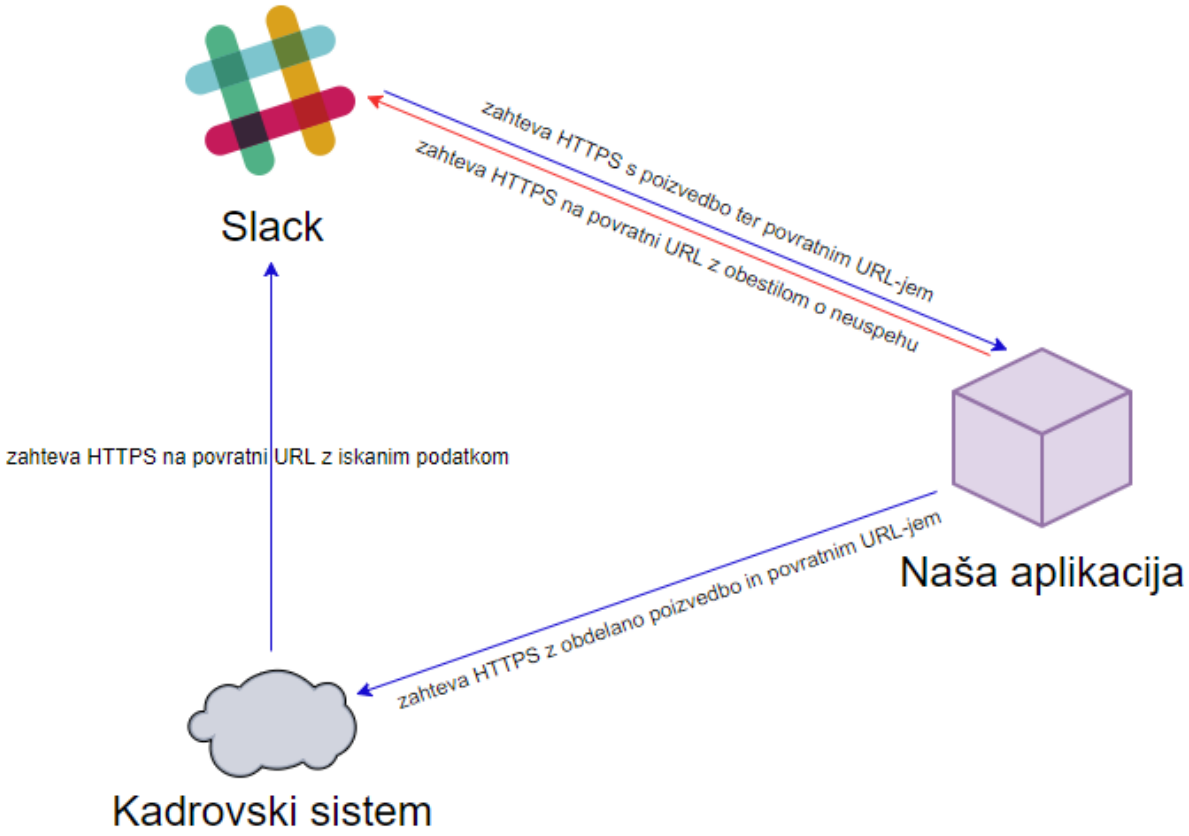
Za vsak podatek, ki ga bomo z našo aplikacijo želeli izpostaviti uporabniku, bo v Apexu potrebno napisati metodo za njegovo poizvedbo. Upoštevati bo potrebno pravice posameznega uporabnika v sistemu. Torej, če nek podatek uporabniku v sistemu ni na voljo, mu ga tudi ne smemo vrniti. Kot smo že omenili, bomo v kadrovski sistem posredovali poizvedbo, kjer bo treba preveriti, kateri podatek išče uporabnik, in nato klicati ustrezno metodo za pridobitev podatka.

3.2 Razvoj arhitekturnega ogrodja aplikacije

Ogrodje naše aplikacije sestavljajo trije členi, ki med seboj komunicirajo s pomočjo protokola HTTPS. Vhod v aplikacijo je implementiran s pomočjo ukazov s poševnico v aplikaciji Slack. Ta uporabniško poizvedbo posreduje naši aplikaciji, ki teče na spletnem strežniku in čaka na zahteve. Ko naša aplikacija prejme zahtevo z uporabniško poizvedbo, nanjo takoj odgovori z informativnim stavkom, potem pa nadaljuje z obdelavo poizvedbe. V primeru, da ne najde pomena poizvedbe, uporabniku to sporoči z zahtevo na povratni URL, ki smo ga dobili ob začetni zahtevi. V nasprotnem primeru pa poizvedbo, skupaj s povratnim URL-jem, posreduje kadrovskemu sistemu. Tam sistem pridobi podatke in jih sporoči uporabniku z zahtevo na povratni URL. Če podatkov v sistemu ni ali uporabnik do njih nima dostopa, mu to sporočimo s sporočilom na povratni URL. Slika 3.1 prikazuje shemo povezav med vsemi členi v arhitekturi.

3.2.1 Slack ukaz s poševnico

Implementacija ukaza s poševnico [26] je potekala brez programiranja in je bila zelo enostavna. Kot administrator okolja Slack smo se prijavili na naslov `https://api.slack.com`. Najprej smo morali izdelati aplikacijo za okolje Slack, nato pa smo lahko izdelali še ukaz s poševnico. Naš ukaz smo definirali



Slika 3.1: Shema povezav. Povezave so prikazane samo v smeri zahtev, odgovorov na posamezno zahtevo pa nismo vključili v shemo. Z rdečo barvo je obarvana povezava, ki se zgodi v primeru, da naša aplikacija ne najde pomena poizvedbe.

kot "/slobot". Poleg tega smo definirali tudi, na kateri naslov URL se ob izvedbi ukaza pošlje poizvedba uporabnika. Ob tem smo dobili verifikacijski žeton za implementacijo znotraj naše aplikacije.

Poleg poizvedbe uporabnika se po izvedbi ukaza v zahtevi HTTP *post* pošljejo še identifikacijski podatki uporabnika, verifikacijski žeton in povratni URL. Pošljejo se tudi ostali podatki, a za nas so najbolj pomembni našteti. Z verifikacijskim žetonom avtenticiramo izvor zahteve na strani naše aplikacije, z identifikacijskimi podatki uporabnika pa prepoznamo, za koga pridobivamo podatke. Ukaz s poševnico po izvedbi na odgovor čaka 3 sekunde. Če odgovor dobi, se ta prikaže uporabniku, v nasprotnem primeru pa uporabniku sporoči

napako. Prav tako ukaz s poševnico še 30 minut na naslovu povratnega URL-ja čaka na morebitne zapoznele odgovore, ki jih ob prejemu prikaže uporabniku.

3.2.2 Naša aplikacija

Najprej smo implementirali enostaven strežnik s knjižnico waitress [6] v Pythonu. Strežnik smo nato nadgradili s knjižnico falcon [9], ki implementira arhitekturo REST. Tako smo zelo na hitro imeli vzpostavljeno okolje, ki lahko sprejema in odgovarja na zahteve. Ob prejemu zahteve *post* preverimo, ali se prejeti verifikacijski žeton ujema s tistim, ki smo ga prejeli ob izdelavi ukaza s poševnico. V primeru, da žeton ni enak, na zahtevo odgovorimo s statusom HTTP 401. V nasprotnem primeru odgovorimo s statusom HTTP 200 ter infromativnim stavkom: "Samo trenutek. Iščem odgovor na vnos.".

Tukaj sledi naša logika obdelave uporabniške poizvedbe. Če naš program ni našel pomena poizvedbe, pošljemo zahtevo na povratni URL s sporočilom o neuspehu. Drugače pa obdelano poizvedbo s povratnim URL-jem posredujemo v kadrovski sistem. Platforma Force.com za dostop v sistem zahteva avtorizacijo s protokolom OAuth 2.0 [23], zato v zahtevi posredujemo tudi dostopni žeton ("access token"). V primeru, ko ta ni več veljaven, platformo zaprosimo za novega z osvežilnim žetonom ("refresh token") in nato ponovno pošljemo prvotno zahtevo. Kako smo pridobili osvežilni žeton, bomo opisali v razdelku 3.2.3. Če je kadrovski sistem na našo zahtevo odgovoril s HTTP statusom 200, zaključimo program, drugače pa na povratni URL pošljemo zahtevo o neuspešnem pridobivanju podatkov.

3.2.3 Komunikacijski vmesnik v kadrovskem sistemu

Preden smo v kadrovskem sistemu sprogramirali pridobivanje podatkov, smo morali v sistemu pridobiti osvežilni žeton za avtorizacijo. Za njegovo pridobitev smo morali v sistemu izdelati *povezano aplikacijo* [25], ki predstavlja naš Pythonov program. Tukaj smo dobili *ključ potrošnika* in *skrivnost potrošnika*,

s katerima smo nato pridobili osvežilni žeton.

V kadrovskem sistemu smo ustvarili razred *SloBotController*, ki ga uporabljamo za pridobivanje podatkov. Da lahko iz zunanjih aplikacij dostopamo do ustvarjenega razreda, smo uporabili Apexove tehnologije REST [25]. Kot prikazuje slika 3.2, smo razredu dodali anotacijo *@RestResource*. Izdelali smo tudi metodo, ki sprejema zahteve HTTP *post*. To smo ji omogočili z anotacijo *@HttpPost*. Tukaj sprejmemo zahtevo, ki smo jo posredovali iz Pythonovega programa. Ko iz sistema pridobimo podatke, jih z zahtevo na povratni URL posredujemo nazaj v Slack. Na osnovno zahtevo, ki je sklicala pridobivanje podatkov, pa vrnemo HTTP status 200.

```
1  @RestResource(urlMapping='/slobot')
2  global with sharing class SloBotController  {
3
4      @HttpPost
5      global static String pridobiPodatke() {
6          // Tukaj pridobimo podatke po
7          // katerih poizveduje uporabnik
8          return 'OK';
9      }
10 }
```

Slika 3.2: Koda Apex, ki sprejema zahteve na URL: "/slobot".

3.3 Razčlenjevanje uporabniških poizvedb z razčlenjevalnikom za slovenski jezik

Poizvedbo smo najprej zapisali v tekstovno datoteko, saj je to vhod, ki ga sprejme razčlenjevalnik. V niz smo zapisali ukaz z vsemi argumenti za pogon razčlenjevalnika ter ga pognali s spodnjo vrstico kode, ki sproži ukaz v podlupini.

```
os.system(ukazniNiz)
```

Ko se razčlenjevanje zaključi, imamo na voljo oblikoslovno označeno in skladenjsko razčlenjeno besedilo v formatu XML-TEI, v datoteki XML, ki smo

jo v ukazu določili kot izhod. Datoteko v program preberemo s knjižnico xml.dom.minidom [11], ki nam omogoča enostavno brskanje po elementih in njihovih atributih.

Ob pogonu razčlenjevalnika smo opazili, da nalaganje modela znanja v pomnilnik traja več kot 3 minute, saj je ta velik več kot 500 MB in za uspešno obdelavo vnosa potrebuje vsaj 4 GB pomnilnika. Če zadostna količina letega ni na voljo, program vrne napako. Ko je model znanja enkrat naložen v pomnilnik, je razčlenjevanje enega stavka takojšnje. Ker je glavni cilj diplomske naloge to, da se uporabnikom prikrajša čas pridobivanja podatkov, je to nesprejemljiva funkcionalnost. Zato smo sklenili, da bomo logiko razčlenjevalnika poizkusili spremeniti.

3.3.1 Predelava razčlenjevalnika

Od avtorjev razčlenjevalnika smo pridobili izvorno kodo le-tega. Arhitekturo smo želeli spremeniti tako, da bi se model znanja v pomnilnik naložil samo enkrat, ob pogonu strežnika, in ne za vsako uporabniško poizvedbo. Ob pregledu kode smo ugotovili, da je to mogoče.

Program smo spremenili tako, da smo ga ločili na del, ki naloži model znanja za razčlenjevanje in nato razčleni vnos, ter na del, ki ga izvedemo ob prejemu uporabniške poizvedbe, kot smo opisali zgoraj. Potrebno je bilo urediti tudi komunikacijo med njima. To smo naredili s knjižnico jetty [12], ki omogoča hitro implementacijo mikrostoritev v programskem jeziku Java. Z njo smo del, ki razčlenjuje, spremenili tako, da ob zagonu v pomnilnik naloži model znanja in nato požene lokalni strežnik, na katerem čaka na zahteve. Ko zahtevo dobi, jo obdela in nazaj sporoči status razčlenjevanja. Drugi del pa smo spremenili tako, da namesto izdelave primerka razčlenjevalnika, ki bi poizvedbo razčlenila, ta na lokalni strežnik posreduje zahtevo za razčlenjevanje.

Tukaj smo morali posodobiti tudi delovanje našega glavnega strežnika tako, da ob zagonu požene prvi zgoraj opisani del.

3.4 Razvoj algoritma za razumevanje razčlenjenih poizvedb

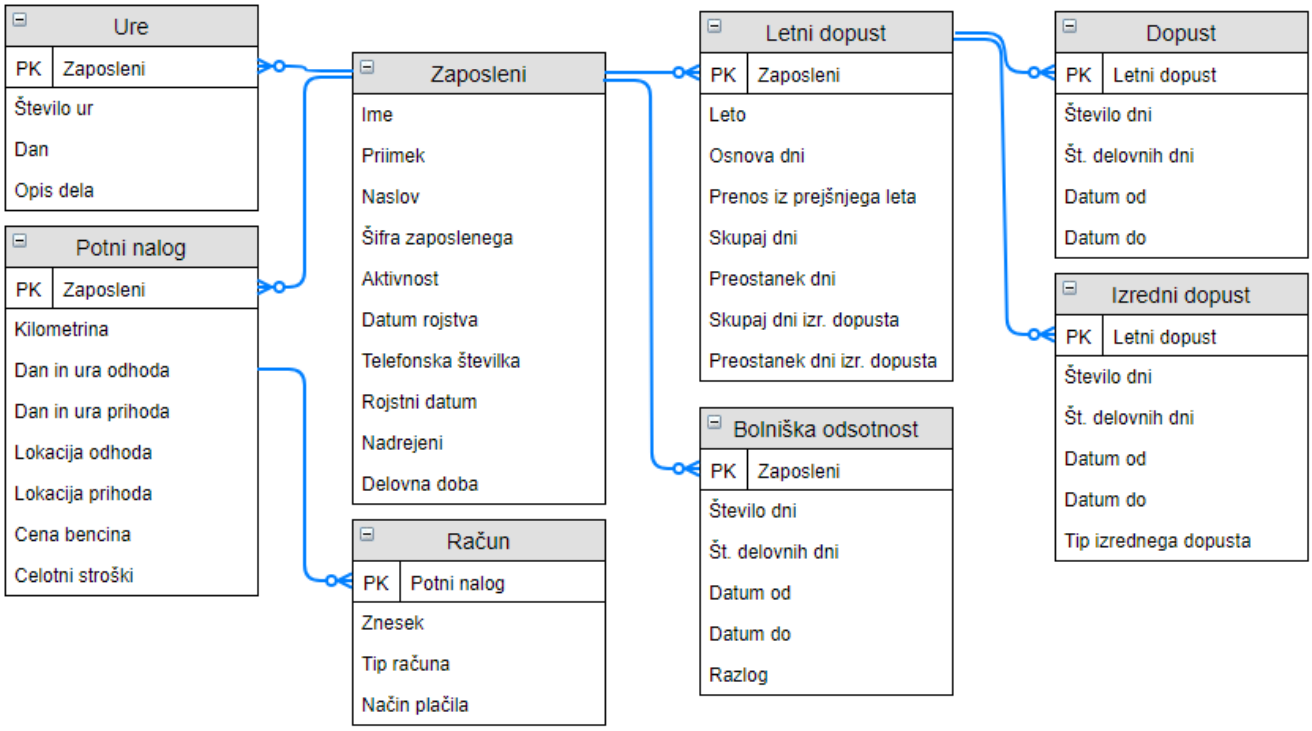
V nadaljevanju bomo opisali, kako smo iz poizvedbe, podane v slovenščini, razbrali, kateri podatek želi pridobiti uporabnik.

3.4.1 Podatki v kadrovskem sistemu

Ker je podatkovni model v kadrovskem sistemu zelo obsežen in tudi ker vsi podatki niso smiselni za povpraševanje, smo si v sklopu diplomske naloge izbrali nekaj tem, po katerih lahko sprašujejo uporabniki. Tematike so prikazane na sliki 3.3. Objekti so med seboj povezani s tipom polja, ki se imenuje *odnos povezave* ("lookup relationship"). V njega se shrani *Id* zapisa, ki ga povezujemo z objektom. Od prikazanih smo najprej začeli razvijati temo o dopustu, do konca implementacije pa smo dodali še temi o izrednem dopustu ter o bolniških odsotnostih. Ostale bi lahko dodali kot nadgradnjo diplomske naloge.

V sklopu posamezne izbrane teme smo določili še, po katerih podatkih lahko sprašujejo uporabniki.

1. Podatki o dopustu:
    - količina preostalega dopusta v trenutnem letu (npr. "Koliko imam še dopusta?"),
    - količina dopusta v preteklih letih (npr. "Koliko počitnic sem imel v letu 2013?"),
    - količina preostalega dopusta nekega drugega uporabnika v trenutnem letu (npr. "Je Janezu Novaku letos preostalo še kaj oddiha?") in
    - količina dopusta nekega drugega uporabnika v preteklih letih (npr. "Koliko dni dopusta je imel Janez Novak v 2015?").



Slika 3.3: Vzorec podatkovnega modela iz kadrovskega sistema.

2. Podatki o izrednem dopustu so podobni podatkom o dopustu, le da tukaj razne sopomenke pri vprašanjih ne pridejo v poštev:

   - količina preostalega izrednega dopusta v trenutnem letu (npr. "Koliko imam še izrednega dopusta?"),
   - količina izrednega dopusta v preteklih letih (npr. "Koliko izrednega dopusta sem imel v letu 2013?"),
   - količina preostalega izrednega dopusta nekega drugega uporabnika v trenutnem letu (npr. "Je Janezu Novaku letos preostalo še kaj izrednega dopusta?") in
   - količina izrednega dopusta nekega drugega uporabnika v preteklih letih (npr. "Koliko dni izrednega dopusta je imel Janez Novak v 2015?").

3. Podatki o bolniški odsotnosti:

- koliko dni je bil uporabnik v nekem časovnem obdobju odsoten zaradi bolezni (npr. “Sem bil julija 2016 kaj bolan?”, “Koliko bolniške sem imel v prejšnjem tednu?”),
- koliko dni je bil nek drug uporabnik v nekem časovnem obdobju odsoten zaradi bolezni (npr. “Koliko bolniškega dopusta je Janez imel maja letos?”, “Koliko je Janez bil septembra bolan?”),
- kdaj je bil uporabnik nazadnje odsoten zaradi bolezni (npr. “Kdaj sem bil nazadnje na bolniški?”) in
- kdaj je bil nek drug uporabnik nazadnje odsoten zaradi bolezni (npr. “Kdaj je imel Janez nazadnje odsotnost zaradi bolezni?”).

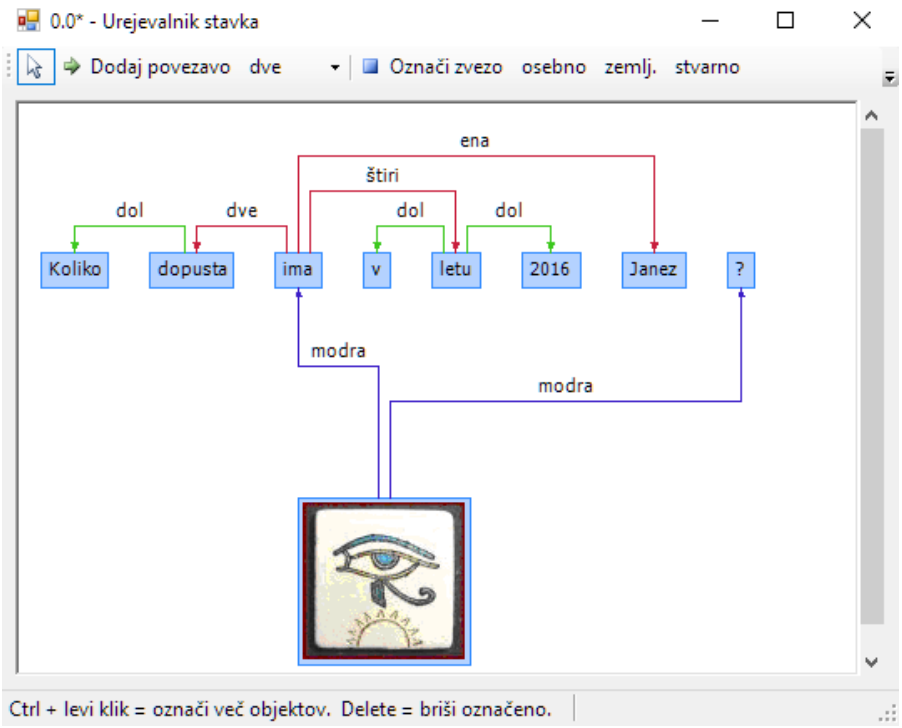
Iz slike 3.3 je jasno, da bi lahko izbrali še več podatkov, a v sklopu diplomske naloge smo se omejili na naštete, ostale pa se lahko doda kot nadgradnjo.

3.4.2 Testna vprašanja in nov učni korpus

Prijatelje in znance smo prosili za pomoč pri pisanju testnih vprašanj. Povedali smo jim, po katerih podatkih lahko sprašujejo, nato pa smo skupaj poizkusili pridobiti čim več primerov vprašanj. Na koncu smo skupaj dobili 858 različnih vprašanj, od tega 615 o dopustu, 204 o izrednem dopustu in 39 o bolniških odsotnostih. Vprašanja smo najprej oblikoslovno in skladenjsko označili z razčlenjevalnikom, nato pa smo jih začeli ročno pregledovati in popravljati s pomočjo programa “Označevalnik stavkov”(slika 3.4). Tega smo dobili v paketu poleg razčlenjevalnika. Pri razčlenjevanju vprašanj smo si pomagali s kazalnikom, ki določa standard za izdelavo učnega korpusa. Kazalnik [13] je nastal v sklopu istega projekta kot Obeliks in razčlenjevalnik.

3.4.3 Iskanje trendov in lastnosti v testnih vprašanjih

Tekom ročnega pregledovanja in popravljanja skladenjskih oznak smo vprašanja grupirali na podlagi podobnosti stavčnih členov in njihovega zaporedja be-



Slika 3.4: Program za ročno skladenjsko označevanje stavkov.

sed. Ko smo končali, smo opazili, da se vprašanja pogosto ponavljajo. Na primer skupina vprašanj:

- "Koliko dopusta imam?",
- "Koliko dopusta še imam?",
- "Koliko oddiha imam?",
- "Koliko neizkoriščenega dopusta še imam?",
- "Koliko neizkoriščenega oddiha še imam?",
- "Koliko počitnic še imam?"

ali:

- "Ali ima Janez to leto dopust?",

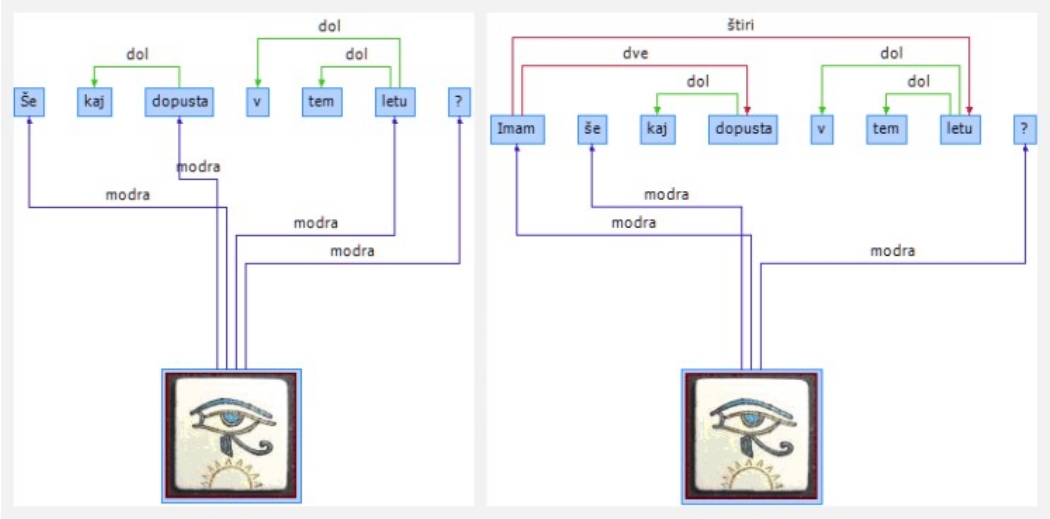
- "Ima Janez letos kaj neizkoriščenega dopusta?",
- "Ima Janez letos še kaj počitnic?".

Našteta vprašanja v posamezni skupini so po strukturi in želenem podatku enaka, loči jih le uporaba različnih izrazov (sopomenk) za besedo "dopust" ter kakšen vrinjen člen ali pridevnik, ki ga lahko ignoriramo. Vseh 858 vprašanj smo tako razporedili v nekaj več kot 70 skupin. Prav tako smo opazili, da se med skupinami veliko stavčnih členov ponavlja oziroma je eden lahko podmnožica drugega. Na primeru zgoraj navedenih skupin imamo ponavljajoči se povedek "imam" oziroma "ima" ter predmet "dopust", ki je podmnožica predmeta "kaj neizkoriščenega dopusta". Tukaj smo ugotovili, da bi bilo še bolje, če bi imeli seznam vseh stavčnih členov, ki se pojavijo v testnih vprašanjih, in da bi bili ti razporejeni v podobne skupine kot zgoraj omenjena primera "imeti" in "dopust". S tem bi lahko še bolj zreducirali in poenostavili možne vnose na skupine in podskupine. To smo v nadaljevanju tudi storili.

Opazili smo tudi pojavljanje vprašanj, ki ne vsebujejo povedka. Na primer:

- "Še kaj dopusta v tem letu?",
- "Koliko dopusta letos?",
- "Oddih?".

Primer skladenjskega označevanja takšnih stavkov je prikazan v levem razdelku na sliki 3.5, v desnem razdelku pa je za primerjavo prikazana razčlenitev enakega stavka s povedkom. V stavkih brez povedka smo opazili, da se povezave drugega nivoja (tabela 2.1), ki med seboj povezujejo stavčne člene, nadomestijo s povezavami tretjega nivoja. Namesto, da povezave izvirajo iz povedka, imajo te izvor v korenu drevesa. Ostale povezave ostanejo enake kot v stavku s povedkom.



Slika 3.5: V levem razdelku je prikazana razčlenitev stavka brez povedka, v desnem pa razčlenitev enakega stavka s povedkom. Pomen prikazanih povezav prikazuje tabela 2.1.

3.4.4 Pretvorba v programske objekte

Tukaj smo prebrano XML datoteko pretvorili v smiselne programske objekte, ki so med seboj povezani. Izdelali smo objekte *Stavek*, *Beseda* in *StavcniClen*. Tako smo si omogočili lažjo obdelavo razčlenjenih vprašanj ter možnost izpisa najdenih stavčnih členov.

Objekt *Beseda*

Konstruktor objekta kot parameter prejme XML element besede v obliki spodnje strukture. Iz njega smo prepisali atribute v atribute objekta.

```
<w lemma="imeti" msd="Ggnspe-n" xml:id="0.0.3">imam</w>
```

Atributi objekta so *mojId* (oz. “xml:id”, ki je unikatni identifikator elementa v celotnem dokumentu), *vrednost* (oz. izvirna oblika besede), *lema* (oz. osnovna oblika besede) ter *msd* (oz. oblikoskladenjska oznaka besede).

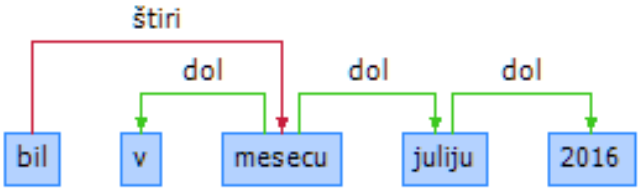
Objekt *Stavek*

Tukaj konstruktor kot parameter prejme XML element stavka, ki ga v strukturi DOM oznanja značka "<s>" (prikazana na sliki 2.3). Iz njegove vsebine napolnimo sledeče atribute objekta:

- atribut *mojId*, ki podobno kot pri objektu *Beseda* vsebuje unikatni identifikator stavka,
- atribut *besede*, ki je tipa *slovar*, pri katerem je ključ atribut *mojId* objekta *Beseda*, na katerega kaže,
- atribut *povezaveOd*, ki je tudi tipa *slovar*, pri katerem je ključ prav tako atribut *mojId* objekta *Beseda*, kaže pa na tabelo vseh XML elementov povezav (značka "<link>"), ki izvirajo iz njega,
- atribut *povedek*, ki je objekt tipa *StavcniClen* in vsebuje povedek stavka (najdemo ga s tipom povezave "modra", ki povezuje koren stavčnega drevesa ter glavni glagol v stavku),
- atribut *osebek*, ki je prav tako objekt tipa *StavcniClen* in vsebuje osebek stavka (najdemo ga s tipom povezave "ena"),
- atribut *predmeti*, ki je tabela, v kateri so vsi najdeni predmeti stavka (najdemo jih s tipom povezave "dve"), ti so tipa *StavcniClen*,
- atribut *dolocila3*, ki je podoben prejšnjemu, le da so v tabeli vsa prislovna določila načina (najdemo jih s tipom povezave "tri"),
- atribut *dolocila4*, ki je enakega tipa kot zgornja dva, le da so v tabeli vsa prislovna določila časa, kraja ter vzroka (najdemo jih s tipom povezave "štiri") in
- atribut *cleni*, ki je tudi enakega tipa kot zgornji trije, napolni pa se z vsemi najdenimi členi v primeru, ko stavku ne najdemo povedka (najdemo jih s tipom povezave "modra").

Objekt *StavcniClen*

Konstruktor ima parametra *stavek* in *beseda*. Prvi je primerek objekta *Stavek*, ki mu pripada, drugi pa je primerek objekta *Beseda*, ki predstavlja jedro tega stavčnega člena. Slednji se prepiše v istoimenski atribut objekta. Ta vsebuje še atribut *mojePovezave*, ki predstavlja tabelo objektov *StavcniClen*. V tej najdemo člene, ki so v stavku z besedo povezani s povezavo prvega nivoja. Tabelo se napolni rekurzivno s pomočjo vhodnega parametra *stavek*, v katerem imamo shranjene vse besede in povezave med njimi. Na primer v stavku: "Koliko sem bil bolan v mesecu juliju 2016?", želimo ustvariti stavčni člen prislovnega določila časa: "v mesecu juliju 2016". V konstruktor posredujemo objekt tega stavka ter objekt glavne besede želenega stavčnega člena, v tem primeru je to objekt besede "mesecu" (slika 3.6 prikazuje povezave v navedenem stavčnem členu). Končni rezultat je v obliki JSON prikazan na sliki 3.7.



Slika 3.6: Povezave med besedami v stavčnem členu: "v mesecu juliju 2016".

3.4.5 Izpis in grupiranje stavčnih členov iz vseh testnih vprašanj

V sklopu novo izdelanih programskih objektov smo za vsakega spisali tudi funkcijo za izpis njegove vsebine. Nastali program smo pognali z vhodom novega učnega korpusa ter izpisali vse stavčne člene, ki se pojavijo. Izpisane stavčne člene smo nato začeli ročno grupirati, kjer smo odstranili vse nepomembne člene ali pridevnike, ki nimajo vpliva na pomen. Grupirali smo tudi

```
StavcniClen: {
    beseda: Beseda: {vrednost: "mesecu",...},
    mojePovezave: [
        StavcniClen: {
            beseda: Beseda: {vrednost: "v",...},
            mojePovezave: []
        },
        StavcniClen: {
            beseda: Beseda: {vrednost: "juliju",...},
            mojePovezave: [
                StavcniClen: {
                    beseda: Beseda: {vrednost: "2016",...},
                    mojePovezave: []
                }
            ]
        }
    ]
}
```

Slika 3.7: Podatkovna struktura objektov *StavcniClen*, ki predstavljajo prislovno določilo časa: "v mesecu juliju 2016".

po sopomenkah (npr. člena "dopust" in "oddih" spadata v isto skupino) in po besedah, ki bi se lahko v stavčnem členu zamenjali in se pomen ne bi dosti spremenil (npr. "v tem letu" in "v prejšnjem letu"). V skupinah smo vse besede stavčnega člena zapisali z lemo. Prav tako smo pri zapisovanju obdržali drevesno strukturo. Sproti smo si zapisovali tudi, v katero temo spada posamezna skupina stavčnih členov. Na primer stavčni člen "imeti", se pojavi v vseh treh temah, medtem ko se stavčni člen "Koliko dni dopusta", pojavi samo pri temi o dopustu. Spodaj bomo prikazali nekaj primerov skupin najbolj pogostih stavčnih členov ter tudi nekaj zanimivih posebnosti. Za vsa testna vprašanja smo na koncu dobili 54 različnih skupin stavčnih členov, med katerimi pa ni osebkov, saj ti ne vsebujejo podatkov o temi in jih nima smisla grupirati.

Povedek

Na primer povedek v stavku: "Koliko prostih dni lahko še izkoristim?", ki se lahko pojavi pri temah o dopustu ter izrednem dopustu. Skupina povedka je prikazana spodaj.

```
izkoristiti -> lahko
```

Predmet

Primer dveh predmetov v stavku: "Je Janezu preostalo še kaj dni izrednega dopusta?". Tukaj imamo predmeta "Janezu" in "kaj dni izrednega dopusta". Slednji se lahko pojavi samo pri temi o izrednem dopustu, prikaz njegove skupine je spodaj.

```
dan -> dopust -> izreden
```

Posebnosti pri predmetih so lastna imena (npr. "Janezu"), saj jih ne gre poenotiti z lemo. Namesto leme smo tukaj uporabili prva dva znaka oblikoskladenjske oznake lastnih imen: "Sl". Prvi znak nam pove, da gre za samostalnik, drugi pa, da gre za lastno ime.

Povezava tipa štiri

Tukaj je dober primer stavek: "Koliko dni izrednega dopusta imam v tem letu?". Tukaj imamo dve prislovni določili časa. Njuni skupini sta prikazani spodaj. Prvo se lahko pojavi pri vseh treh temah, drugo pa samo pri temi o izrednem dopustu.

```
dan -> dopust -> izreden
    -> koliko


leto -> v
     -> ta
```

Posebnost pri prislovnih določilih časa so členi, ki vsebujejo letnice (npr. "v 2016"). Tudi tukaj smo namesto leme uporabili oblikoskladenjsko oznako. Ta je: "Kag".

Posebnost so tudi imena mesecev (npr. "februarja letos"). Ker nismo želeli imeti posamezne skupine stavčnega člena za vsak mesec, smo si izmislili univerzalni niz: "{!imeMeseca}".

3.4.6 Iskanje teme uporabniške poizvedbe

Pridobljene skupine stavčnih členov smo se odločili vključiti v program z namenom, da bi nam te pomagale pri iskanju teme posamezne poizvedbe. Vsako skupino smo predstavili s primerkom objekta *StavcniClen*. Tega smo nato kot ključ vstavili v *slovar*, kjer vsak kaže na tabelo vseh tem, pri katerih se skupina lahko pojavi. Za lažjo predstavo je na sliki 3.8 shematični prikaz slovarja.

```
slovar: {
    StavcniClen(preostati -> biti): ["izredni dopust", "dopust"],
    StavcniClen(dopust -> bolniški): ["bolniška"],
    ...
}
```

Slika 3.8: Vzorec slovarja, v katerem so ključi posamezne skupine. Te so predstavljene z objektom *StavcniClen*, vsaka pa kaže na tabelo tem, v katerih se lahko pojavi.

Slovar smo nato razdelili na štiri dele, vsak predstavlja po en tip stavčnega člena (brez osebkov).

Ko imamo za posamezno razčlenjeno poizvedbo izdelan objekt *Stavek*, nadaljujemo z iskanjem teme. Iskanje teme smo ločili na dva tipa. Prvi tip je za stavke s povedkom, drugi pa je za stavke brez povedka. Pri prvem vsak stavčni člen objekta primerjamo s primernim tipom skupin. Pri drugem pa vse najdene stavčne člene iz atributa *cleni* primerjamo z vsemi tipi skupin razen s skupinami povedkov. Pri primerjavi s posamezno skupino upoštevamo drevesno strukturo ter leme besed. Med vsemi skupinami izberemo tisto, ki se s stavčnim členom ujema v največ možnih vozliščih. Za vsak stavčni člen, ki smo mu našli skupino, si shranimo množico tem, v katere lahko spada. Na koncu nad danimi množicami naredimo presek in tako dobimo temo, v katero spada stavek. V primeru, da je v preseku več kot ena tema, uporabniku sporočimo, da naj poizvedbo poizkusi napisati na drugačen način.

3.4.7 Sopomenke oz. možne nadomestne besede

Ker smo pri grupiranju stavčnih členov upoštevali možne sopomenke, je to potrebno narediti tudi pri posameznih besedah v stavku poizvedbe. Zato smo nadgradili objekt *Beseda* in mu dodali atribut *sinonim*. Za vsako besedo, ki se pojavi v skupinah stavčnih členov, smo poiskali seznam možnih sopomenk oz. besed, ki bi jo lahko zamenjale in s tem ne bi kaj dosti spremenile pomena stavčnega člena. S tem smo nato vsaki besedi ob izdelavi pripisali možno zamenjavo. V ta sklop besed smo dodali tudi imena vseh mesecev, ki jih nadomesti prej omenjeni niz: "{!imeMeseca}". Seznam vseh je prikazan na sliki 3.9. Tako smo izboljšali učinkovitost iskanja teme poizvedbe.

```
dopust: [oddih, počitnice],
izkoristiti: [porabiti, uporabiti, izrabiti, vnovčiti],
preostati: [ostati],
volja: [razpolaga],
dan: [ura, minuta, sekunda],
letos: [lani, predlani],
ta: [prejšnji, trenuten, tekoč, sedanji, predhoden, pretekel,
    bivši, lanski, letošnji, predlanski],
{!imeMeseca}: [januar, februar, marec, april, maj, junij,
    julij, avgust, september, oktober, november, december]
```

Slika 3.9: Na sliki je prikazan seznam vseh besed, ki se pojavijo v skupinah stavčnih členov (in imajo hkrati možne sopomenke oz. možne zamenjave v stavčnem členu). Besede kažejo na svoje možne sopomenke oz. na besede, ki jih lahko v stavčnem členu zamenjajo.

3.4.8 Iskanje podatka znotraj posamezne teme

Pri vseh treh temah lahko uporabniki sprašujejo po nekem podatku o sebi ali o nekom drugem ter po časovnem obdobju tega podatka. Zato smo dodali logiko za iskanje oseb ter časovnih obdobij. Ko opravimo iskanje oseb ter časovnih obdobij, izvemo točno, kateri podatek išče uporabnik. Če nismo našli nobene osebe, uporabnik verjetno išče podatke o sebi, in če nismo našli nobenega časovnega obdobja, verjetno išče podatke o trenutnem letu. V nadaljevanju bomo opisali še logiki za iskanje oseb ter časovnih obdobij.

Iskanje oseb

Tukaj logiko ponovno ločimo na stavke s povedkom ter na stavke brez povedka. Kadar stavek ima povedek, najprej preverimo če ima osebek. Če ga ima, v njem poiščemo vse besede, ki imajo oblikoskladenjsko oznako lastnega imena ("Sl"). Če ga nima, ali če ne vsebuje nobene besede z iskano oblikoskladenjsko oznako, nadaljujemo iskanje med predmeti stavka, saj smo samo pri njih med testnimi vprašanji opazili uporabo lastnih imen. Pri vsakem predmetu ponovimo iskanje vseh besed z oblikoskladenjsko oznako lastnega imena. Ko je stavek brez povedka, iskanje poteka enako, le da iščemo med vsemi najdenimi členi, saj nimamo podatka o tem, kakšen tip stavčnega člena so. Če po tem postopku ne najdemo nobenih lastnih imen, smatramo, da uporabnik išče podatek o sebi.

Iskanje časovnih obdobij

Tudi tukaj logiko ločimo na stavke brez ter s povedkom in tudi tukaj je logika iskanja v obeh primerih enaka, razlikuje se samo po tem, med katerimi členi iščemo podatek. V primeru brez povedka iščemo med vsemi najdenimi členi, v nasprotnem primeru, pa iščemo samo med členi, ki so na povedek povezani s povezavo tipa štiri. Ta vsebuje prislovna določila kraja, vzroka ter časa. Slednji so tisti, v katerih najdemo iskane podatke. Iskanje poteka tako, da člene primerjamo s skupinami stavčnih členov, ki nosijo podatek o časovnem obdobju (primeri teh skupin so prikazani na sliki 3.10). Ko najdemo ujemajoči par, smo našli tudi iskane podatke, le razbrati jih je še potrebno ven. V stavčnem členu nato poiščemo ali gre za leto, mesec ali teden, ter za katero leto, mesec ali teden gre. Pri slednjih iščemo ključne besede tipa trenutni, prejšnji, lanski itn. Pri mesecih poleg tega iščemo še po njihovih imenih, pri letih pa po oblikoskladenjski oznaki za števila ("Kag"). Možne so seveda tudi kombinacije vseh (npr. "v aprilu 2014"). Če ne najdemo ničesar, privzamemo, da uporabnik išče podatke za trenutno leto.

```
1  letos -> {!imeMeseca}        (npr. letos aprila)
2
3  leto -> ta                   (npr. v tem letu)
4       -> v
5
6  leto -> Kag                  (npr. leta 2016)
7
8  {!imeMeseca} -> Kag          (npr. junija 2013)
9  ...
```

Slika 3.10: Primeri skupin stavčnih členov, ki nosijo podatek o časovnem obdobju. Z njimi primerjamo stavčne člene, najdene v uporabniški poizvedbi, ki so na povedek povezani s povezavo tipa štiri.

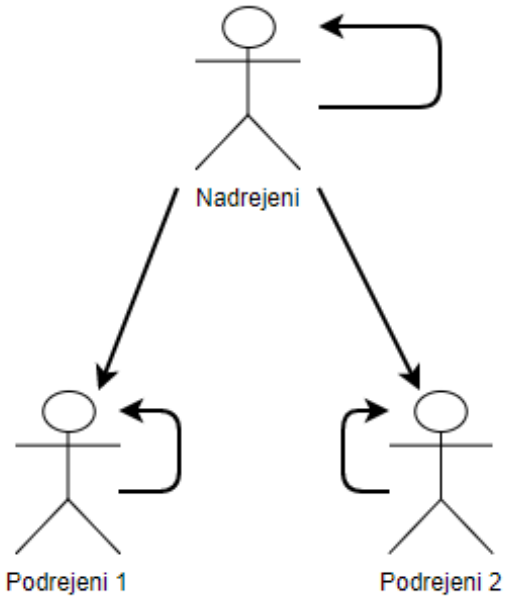
3.5 Pridobivanje podatkov v kadrovskem sistemu

V kadrovski sistem preko zgoraj opisanega vmesnika prejmemo zahtevo HTTP *post*. Ta vsebuje podatke o uporabniku ter o njegovi poizvedbi. Najprej preverimo, če uporabnika imamo v sistemu. Če ga najdemo, nadaljujemo s pridobivanjem podatkov. Za vsak podatek, po katerem lahko sprašuje uporabnik, smo naredili metodo, ki tega pridobi. Iz zahteve preberemo, kateri podatek išče uporabnik ter program preusmerimo v ustrezno metodo, poleg pa posredujemo še podatka o morebitnem drugem uporabniku ter o morebitnem časovnem obdobju. V posamezni metodi skušamo pridobiti iskani podatek. Če podatek najdemo, preverimo še, ali ima uporabnik dostop do zapisa, ki nosi ta podatek. Komunikacijskemu vmesniku v kadrovskem sistemu nato vrnemo najdene podatke, ta pa jih posreduje nazaj uporabniku.

3.5.1 Pravice v kadrovskem sistemu

Platforma Force.com omogoča enostavno vodenje dostopov do podatkov. V našem kadrovskem sistemu imamo sledeča pravila o podatkih, ki jih uporabniki lahko iščejo. Uporabnik za vse podrejene vidi vse podatke o dopustih, izrednih dopustih ter o bolniških odsotnostih. Uporabnik za svoje nadrejene ter zaposlene, ki so na enakem nivoju kot on, nima dostopa do njihovih

podatkov. Primer sheme pravic je prikazan na sliki 3.11, kjer imamo nadrejenega in dva podrejena. Podrejena tako lahko sprašujeta samo po svojih podatkih, nadrejeni pa po svojih ter po podatkih obeh podrejenih.



Slika 3.11: Slika s puščicami prikazuje, kakšne pravice imajo uporabniki.

3.5.2 Primer metode za poizvedovanje po podatku

Na sliki 3.12 je prikazana metoda, ki v kadrovskem sistemu poišče podatke o dopustu nekega uporabnika v določenem letu. V metodi sta dve poizvedbi SOQL, ena za pridobivanje iskanega podatka, druga pa za pridobivanje podatka o pravicah uporabnika za najden podatek.

```
1  String dneviDopustaUporabnikaVLetu(User uporabnik,
2                                     String imeOsebe,
3                                     String leto) {
4      Letni_dopust ld = [SELECT Id, Skupaj_dni
5                         FROM Letni_dopust
6                         WHERE Leto=:leto
7                         AND Zaposleni.Name=:imeOsebe
8                         LIMIT 1];
9      String odgovor = "";
10     if (ld != null && ld.Skupaj_dni != null) {
11         // iskanje pravic
12         UserRecordAccess dostop = [SELECT HasReadAccess
13                                    FROM UserRecordAccess
14                                    WHERE UserId=:uporabnik.Id
15                                    AND RecordId=:ld.Id];
16         if (dostop.HasReadAccess) {
17             // Ima dostop, lahko odgovorimo z iskanim podatkom
18             odgovor = "Zaposleni "+imeOsebe+" je imel v letu "+
19                       leto+" "+ld.Skupaj_dni+" dni dopusta.";
20         } else {
21             // Nima dostopa
22             odgovor = "Nimate pravic do željenega podatka."
23         }
24     } else {
25         // Odgovor, ko v sistemu ni iskanega podatka
26         odgovor = "V sistemu ni zapisa o dopustu "+
27                   imeOsebe+" iz leta "+leto+".";
28     }
29     return odgovor;
30 }
```

Slika 3.12: Metoda na sliki je namenjena pridobivanju podatka o dopustu nekega uprabnika v določenem letu. Parametri metode so objekt uporabnika, ki sprašuje, ime uporabnika, za katerega išče podatek, in leto, za katero ga ta podatek zanima. V prvi poizvedbi SOQL na vrstici 4 poiščemo zapis, ki vsebuje iskani podatek. Če smo zapis našli, naredimo še drugo poizvedbo SOQL na vrstici 12, kjer poiščemo, ali ima uporabnik, ki sprašuje, pravice za dostop do zapisa, najdenega v prvi poizvedbi SOQL.

Poglavje 4

Testiranje in evalvacija

4.1 Opis testiranja

Program za razumevanje uporabniških poizvedb smo tekom razvoja ves čas testirali s pomočjo našega novega oblikoslovno ter skladenjsko označenega učnega korpusa ter brez uporabe razčlenjevalnika. Ko smo z razvojem končali, je naš program za vsa ročno pregledana vprašanja v učnem korpusu pravilno našel podatke, po katerih sprašujejo. Potem smo v program vključili še razčlenjevanje z razčlenjevalnikom. Za preverjanje uspešnosti programa smo mu dodali razred *PreveriUspesnost*, v katerem smo napisali logiko za primerjavo dveh vhodnih XML datotek. V datotekah se pričakujejo enaki stavki v enakem zaporedju. Eno izmed njiju določimo kot pravilno ter jo primerjamo z drugo. Razred stavke v parih pretvori v objekt *Stavek* in za vsakega ugotovi, po katerem podatku sprašuje. Na koncu pare obdelanih stavkov primerjamo. Tukaj nismo preverjali samo uspešnosti naše logike za iskanje pomena, ampak tudi uspešnost razčlenjevanja. Testna vprašanja smo dali na vhod razčlenjevalniku in jih razčlenjene, s pomočjo novega razreda, primerjali z ročno razčlenjenim učnim korpusom. Rezultati so predstavljeni v nadaljevanju.

4.1.1 Rezultati primerjave

V tabeli 4.1 so prikazani rezultati primerjave stavkov iz ročno razčlenjene ter avtomatsko razčlenjene datoteke. Uspešnost našega programa se tako z uporabo razčlenjevalnika in modela znanja naučenega na skladenjsko označenem delu učnega korpusa ssj500k zmanjša iz 100 % na približno 75 %. Razlog za to je precej nizek odstotek pravilnih stavčnih členov, saj se pri iskanju pomena vprašanja precej zanašamo na njih. Tukaj smo se vprašali, zakaj je odstotek pravilnih stavčnih členov tako nizek. Predvidevali smo, da je to zaradi slabega zastopanja vprašanj v skladenjsko označenem delu učnega korpusa ssj500k, iz katerega je zgrajen model znanja, s katerim razčlenjujemo. V njem je namreč od 11.411 skladenjsko označenih stavkov samo 368 vprašanj. Ker so vprašanja po strukturi nekoliko drugačna od navadnih stavkov, se nam je predpostavka zdela smiselna. Odločili smo se jo tudi testirati.

	Št. vseh	**Št. pravilnih**	**Odstotek pravilnih**
Povezave med besedami	5933	5497	92.65 %
Stavčni členi	2855	1963	68.76 %
Iskanje pomena stavkom	858	647	75.41 %

Tabela 4.1: Rezulati primerjave obdelave in razčlenjevanja našega učnega korpusa in testnih stavkov, razčlenjenih z razčlenjevalnikom. Tukaj smo za razčlenjevanje testnih stavkov uporabili model znanja, zgrajen z učnim korpusom ssj500k. Uspešnost našega programa je prikazana v zadnji vrstici tabele.

Ker smo tekom razvoja naše rešitve ustvarili skladenjsko označen učni korpus, ki vsebuje 858 vprašanj, smo se odločili, da ga združimo s skladenjsko označenim delom učnega korpusa ssj500k. Skupni učni korpus tako vsebuje 12.269 skladenjsko označenih stavkov, od katerih je 1.226 vprašanj, kar je že dobrih 10 %. Z njim ter z razčlenjevalnikom smo zgradili nov model znanja. Ponovno smo pognali program za primerjavo, a tokrat smo za avtomatsko razčlenjanje uporabili nov model znanja. Rezultati nove primerjave so prikazani v tabeli 4.2. Ti so potrdili našo predpostavko. Odstotek pravilnih stavčnih členov se je dvignil z 68.76 % na 76.11 %, odstotek uspešnosti

našega programa pa z 75.41 % na 84.03 %.

	Št. vseh	**Št. pravilnih**	**Odstotek pravilnih**
Povezave med besedami	5933	5628	94.86 %
Stavčni členi	2855	2173	76.11 %
Iskanje pomena stavkom	858	721	84.03 %

Tabela 4.2: Rezulati primerjave obdelave in razčlenjevanja našega učnega korpusa in testnih stavkov označenih z razčlenjevalnikom. Tukaj smo za razčlenjevanje testnih stavkov uporabili na novo zgrajeni model znanja. Uspešnost našega programa je prikazana v zadnji vrstici tabele.

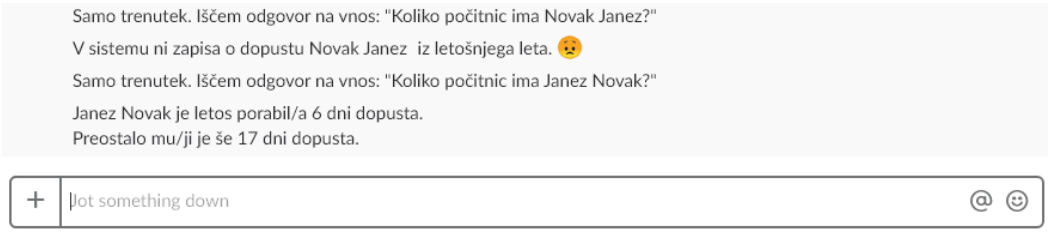
4.2 Evalvacija

Z rezultati uspešnosti programa, ki išče pomen v uporabniških poizvedbah podanih s slovenščino, smo zadovoljni. Program smo razvili tako, da lahko z rastočim številom testnih vprašanj izboljšamo uspešnost razčlenjevanja poizvedb in posledično tudi njegovo uspešnost. To je dober nastavek za nadaljnje delo. A poleg tega, ima aplikacija kot celota tudi nekaj slabosti, ki jih bomo predstavili v nadaljevanju.

4.2.1 Slabosti končne aplikacije

Glavna slabost naše končne aplikacije je definitivno povprečni čas odgovora, ki znaša 19 sekund. Razlog za to je v oblikoslovnem označevalniku Obeliks. Podobno kot razčlenjevalnik mora tudi Obeliks za obdelavo poizvedbe v pomnilnik naložiti po en model znanja za operaciji oblikoskladenjskega označevanja ter lematizacije. Čeprav sta ta dva modela znatno manjša od tistega, ki ga uporabljamo pri razčlenjevanju, je ta čas še vseeno prevelik.

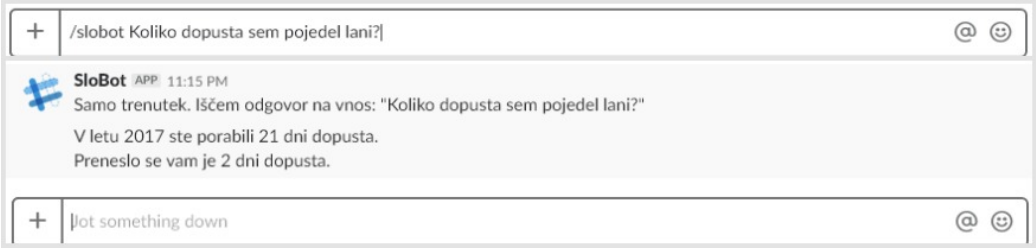
Slabost je zagotovo tudi površno filtriranje po imenih pri pridobivanju zapisov v kadrovskem sistemu. Tam najdeno ime v poizvedbi iščemo v takšni obliki, kot ga je zapisal uporabnik. Primer je prikazan na sliki 4.1.



Slika 4.1: Na sliki sta prikazana primera enake poizvedbe z zamenjanim vrstnim redom imena in priimka iskanega uporabnika. V enem primeru kadrovski sistem najde podatek, v drugem pa ne.

V kadrovskem sistemu smo prav tako površno pripravili odgovore za uporabnika, saj ti ne sklanjajo nobenih besed, ki bi jih bilo potrebno. Primeri so prikazani na slikah 4.1 in 4.2.

Najbolj zanimiva neprimerna funkcionalnost je uspešna obdelava nesmiselnih vnosov. V programu za prepoznavanje pomena ne preverjamo povedkov ter njihovega pomena. Tako lahko uporabnik napiše del stavka, ki vsebuje nek pričakovan stavčni člen, v povedek pa lahko poljubno napiše nekaj neprimernega in na to vseeno dobi odgovor s podatki (slika 4.2).



Slika 4.2: Na sliki je prikazan primer nesmiselne poizvedbe, ki jo sistem kljub temu uspešno obdela in uporabniku vrne odgovor s podatki.

Poglavje 5

Zaključek

Izdelali smo inteligentnega pogovornega agenta (robota) za pridobivanje podatkov iz kadrovskega sistema. Narejena aplikacija v sistemu Slack zajame uporabniško poizvedbo, podano s slovenščino. To s protokolom HTTPS posreduje programu za razčlenjevanje. Tam jo program obdela ter zanjo ugotovi, po katerem podatku sprašuje. Obdelano poizvedbo nato posredujemo v kadrovski sistem, kjer se izvede zadnji korak cikla poizvedbe. To je pridobivanje iskanih podatkov ter vrnitev odgovora s podatki nazaj k uporabniku.

Kljub počasni povprečni hitrosti odgovora uporabniku, je ta čas še vendarle primerljiv s časom iskanja podatkov po sistemu z uporabo brskalnika. V nekaterih primerih je hitrejši, v nekaterih pa ne. To je odvisno od hitrosti internetne povezave, ter od kompleksnosti podatka, ki ga uporabnik išče. Kompleksnejši kot je podatek bolj se mora uporabnik ukvarjati s filtriranjem in iskanjem le-tega. Naša aplikacija je definitivno boljša za uporabo preko mobilne naprave, saj je tam kadrovski sistem precej okrnjen. Slack pa je dovršen na vseh platformah in je tako poznan in pogosto uporabljan vmesnik za uporabnike.

Našo aplikacijo bi lahko nadgradili na veliko področjih. Za uporabnike najbolj uporabna nadgradnja je sigurno izboljšanje časa povprečnega odgo-

vora. Podobno kot smo predelali razčlenjevalnik, bi morali tudi označevalnik Obeliks. Za boljšo uporabnost bi bilo potrebno razširiti nabor možnih podatkov, po katerih lahko poizvedujejo uporabniki. Potrebno bi bilo izboljšati tudi pridobivanje podatkov v kadrovskem sistemu, kjer je v filter vključeno ime nekega uporabnika. Izboljša se lahko tudi sinteza odgovora uporabnikom. Čeprav možnost spraševanja nesmiselnih vprašanj nikogar ne moti in se jih takšnih tudi ne pričakuje, bi bilo potrebno tukaj dodati logiko za prepoznavanje pomena povedka. Potrebno bi bilo tudi shranjevati vse možne uporabniške poizvedbe, saj bi z njimi lahko še izboljšali razčlenjevanje ter iskanje pomena.

Literatura

[1] Kamnik Amebis, d.o.o. Sporazumevanje v slovenskem jeziku. Dosegljivo: http://www.slovenscina.eu/tehnologije, 2012. [Dostopano 28. 2. 2018].

[2] Kamnik Amebis, d.o.o. Klepec. Dosegljivo: https://www.amebis.si/, 2018. [Dostopano 9. 2. 2018].

[3] ChatterBot. Chatterbot in python. Dosegljivo: https://chatterbot.readthedocs.io/en/stable/, 2018. [Dostopano 28. 2. 2018].

[4] clarin.si. Kaj je clarin.si? Dosegljivo: http://www.clarin.si/info/o-projektu/splosne-informacije/, 2017. [Dostopano 10. 2. 2018].

[5] TEI Consortium. Tei: P5 guidelines. Dosegljivo: http://www.tei-c.org/Guidelines/P5/, 2017. [Dostopano 28. 2. 2018].

[6] Agendaless Consulting. Waitress. Dosegljivo: https://docs.pylonsproject.org/projects/waitress/en/latest/, 2017. [Dostopano 10. 2. 2018].

[7] E-računovodstvo. Vida virtualna davčna asistentka. Dosegljivo: http://www.eracunovodstvo.org/blog/pripomocki/vida-virtualna-davcna-asistentka/, 2009. [Dostopano 19. 6. 2017].

[8] Tomaž Erjavec, Darja Fišer, Simon Krek, and Nina Ledinek. The jos linguistically tagged corpus of slovene. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan

Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[9] falcon.org. Falcon. Dosegljivo: `https://falconframework.org/`, 2018. [Dostopano 10. 2. 2018].

[10] National Science Foundation. Minimum-spanning tree parser. Dosegljivo: `http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html`, 2006. [Dostopano 20. 6. 2017].

[11] Python Software Foundation. xml.dom.minidom. Dosegljivo: `https://docs.python.org/2/library/xml.dom.minidom.html`, 2018. [Dostopano 10. 2. 2018].

[12] The Eclipse Foundation. Jetty - servlet engine and http server. Dosegljivo: `https://www.eclipse.org/jetty/`, 2016. [Dostopano 9. 2. 2018].

[13] Peter Holozan, Simon Krek, Matej Pivec, Simon Rigač, Simon Rozman, and Aleš Velušček. Kazalnik 2 - projekt "sporazumevanje v slovenskem jeziku" - specifikacije za učni korpus. Dosegljivo: `http://projekt.slovenscina.eu/Vsebine/Sl/Kazalniki/K2.aspx`, 2008. [Dostopano 20. 2. 2018].

[14] Amazon.com Inc. Amazon alexa. Dosegljivo: `https://developer.amazon.com/alexa`, 2018. [Dostopano 28. 2. 2018].

[15] Apple Inc. Siri. Dosegljivo: `https://www.apple.com/ios/siri/`, 2018. [Dostopano 28. 2. 2018].

[16] Andrej Jugovic. *Abstraktivno povzemanje dokumentov v slovenskem jeziku*. PhD thesis, Univerza v Ljubljani, 2016.

[17] Matjaž Juršič. *Implementacija učinkovitega sistema za gradjno, uporabo in evaluacijo lematizatorjev tipa RDR*. PhD thesis, Univerza v Ljubljani, 2007.

[18] Aleksander Ključevšek. *Statistična analiza slovenskih jezikovnih korpusov*. PhD thesis, Univerza v Ljubljani, 2016.

[19] Simon Krek. *Slovenski jezik v digitalni dobi – The Slovene Language in the Digital Age*. META-NET White Paper Series. Georg Rehm and Hans Uszkoreit (Series Editors). Springer, 2012. Available online at `http://www.meta-net.eu/whitepapers`.

[20] Matej Martinc. *Učinkovito procesiranje naravnega jezika s Pythonom*. PhD thesis, Univerza v Ljubljani, 2015.

[21] Microsoft. Cortana. Dosegljivo: `https://support.microsoft.com/en-us/help/17214/windows-10-what-is`, 2017. [Dostopano 28. 2. 2018].

[22] NLTK. Knjižnica nltk. Dosegljivo: `http://www.nltk.org`, 2017. [Dostopano 8. 2. 2018].

[23] OAuth. Oauth 2.0. Dosegljivo: `https://oauth.net/2/`, 2017. [Dostopano 10. 2. 2018].

[24] Inc salesforce.com. Force.com platform. Dosegljivo: `https://developer.salesforce.com/platform/force.com`, 2017. [Dostopano 10. 2. 2018].

[25] Inc Salesforce.com. The technologies behind a lightning platform app. Dosegljivo: `https://developer.salesforce.com/docs/atlas.en-us.fundamentals.meta/fundamentals/adg_intro_tech.htm`, 2017. [Dostopano 10. 2. 2018].

[26] Slack. Slack api. Dosegljivo: `https://api.slack.com/`, 2018. [Dostopano 10. 2. 2018].

[27] Artificial solutions. Teneo. Dosegljivo: https://www.artificial-solutions.com/teneo, 2017. [Dostopano 20. 6. 2017].

[28] Telekom. Tia – telekomova interaktivna asistentka. Dosegljivo: http://www.telekom.si/pomoc-in-podpora/tia#, 2009. [Dostopano 19. 6. 2017].