UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Luka Milošević

RAZVOJ APLIKACIJE ZA INTEGRACIJU PODATAKA IZ RAZLIČITIH IZVORA O POVEZANOSTI GENA I BOLESTI

master rad

Beograd, 2022.

Mentor:

dr Jovana Kovačević, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Gordana Pavlović-Lažetić, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

dr Radoslav Davidović, naučni saradnik
Univerzitet u Beogradu, INN "Vinča"

Datum odbrane: _____

Najveću zahvalnost dugujem svojoj mentorki, doc. dr Jovani Kovačević, za sve savete i veliko strpljenje tokom izrade ovog rada. Zahvaljujem se i članovima komisije, prof. dr Gordani Pavlović-Lažetić na izuzetno korisnim komentarima i savetima, kao i dr Radoslavu Davidoviću na predloženoj temi i dragocenim smernicama u vezi sa biološkim detaljima istraživanja. Konačno, na neizmernoj podršci se zahvaljujem svojoj porodici i prijateljima koji su me bodrili tokom studija kao i tokom izrade rada.

Naslov master rada: Razvoj aplikacije za integraciju podataka iz različitih izvora o povezanosti gena i bolesti

Rezime:

Mnogobrojne su baze podataka koje sadrže informacije o povezanosti gena i konkretnih patoloških stanja. Pored velikog značaja u istraživanjima, ove baze pokazuju i neke nedostatke, među kojima je i korišćenje različitih identifikatora za gene i bolesti, kao i to da različite baze koriste različite sisteme anotacija koji nisu međusobno kompatibilni, što otežava poređenje informacija između baza.

Jedinstvena anotacijska datoteka (eng. *annotation file*) sa informacijama o povezanosti gena i bolesti, kao i dodatnim informacijama o genima i bolestima iz različitih izvora pomogla bi i olakšala analizu ovih podataka. U tu svrhu razvijena je aplikacija `GDA` (eng. *Gene Disease Annotation*) čija je implementacija i korišćenje opisano u ovom radu. Aplikacija ima za cilj kreiranje i prikaz anotacijske datoteke u vidu interaktivne tabele koja pruža različite funkcionalnosti. Neke od funkcionalnosti su mogućnost pretraživanja sadržaja po određenom kriterijumu, sortiranja, izvoza u odabrani format, itd.

Ključna informacija sadržana unutar anotacijske datoteke jeste veza gen-bolest. Ova informacija prikuplja se iz besplatnih i javno dostupnih specijalizovanih baza podataka o vezi gen-bolest: `DisGeNet`, `COSMIC`, `HumsaVar`, `Orphanet`, `ClinVar`, `HPO` i `Diseases`. Pored ovih baza, dodatne informacije o genima i bolestima preuzimaju se iz sledećih baza: `UniProt`, `HUGO`, `OBO`, `RGD`, `Ensembl` i `Orphanet Xref`. Poseban izazov prilikom izrade anotacijske datoteke bilo je pronalaženje specifičnih identifikatora za bolest (takozvanih `DOID`) na osnovu naziva bolesti koji su dati opisno, nisu jedinstveni i razlikuju se u različitim bazama.

Tokom izrade aplikacije korišćene su različite metode istraživanja teksta (eng. *text mining*). Aplikacija `GDA` kreirana je u formi veb aplikacije otvorenog koda i biće javno dostupna kao dokerizovana aplikacija za operativne sisteme `Windows`, `MacOS` i `Linux`. Za implementaciju aplikacije korišćen je programski jezik `Python`, razvojni okvir `Django`, dodatak `DataTables` biblioteke `JQuery` programskog jezika `JavaScript`. Određeni delovi procesa kreiranja anotacijske datoteke implementirani su korišćenjem niti radi postizanja boljih performansi izvršavanja.

Nakon završenog procesa kreiranja anotacijske datoteke, broj veza gen-bolest koje ova datoteka sadrži je 294971, gde je čak 97% pronađenih atributa `DOID` tačno.

Ključne reči: gen, bolest, podaci, povezanost, anotacijski, aplikacija, GDA

Sadržaj

Glava 1

Uvod

Razumevanje genske osnove patofizioloških procesa u ljudskom organizmu ima veliki klinički značaj kako u razvoju novih dijagnostičkih pristupa u otkrivanju bolesti tako i u razvoju novih terapijskih modaliteta. Tehnološki napredak u poslednje dve decenije omogućio je nastanak novih metoda molekularne biologije koje su donele značajne prodore u razumevanju genetičke etimologije mnogih bolesti.

Fokus ovog rada je kreiranje jedinstvene anotacijske datoteke sa informacijama o povezanosti gena i bolesti, i dodatnim informacijama o genima i bolestima iz različitih izvora. Relevantni podaci potiču iz velikog broja raznovrsnih eksperimenata, izvedenih u okviru različitih institucija i od strane različitih stučnjaka (genetičari, molekularni biolozi, itd.), što otežava dobijanje sveobuhvatnog pregleda o tome koji su geni uključeni u koje bolesti. Međutim, zbog intenzivnih istraživanja koja se obavljaju na ovu temu, biomedicinska literatura o povezanosti gena i bolesti je veoma bogata. Izdvajanje veza bolest - gen iz različitih baza predstavlja krajnji izazov za čije rešenje se nameće tehnika istraživanja teksta (eng. *text mining*) koja bi predstavljala jednu od glavnih tehnika koje bi se koristile pri kreiranju anotacijske datoteke [1, 7, 22, 23]. U tu svrhu razvijen je softver pod nazivom `GDA` (eng. *Gene Disease Annotation*) otvorenog koda za integraciju podataka iz odgovarajućih baza o povezanosti gena i bolesti i sam prikaz anotacijske tabele.

Kako je sama oblast široka, postoji i veliki broj različitih baza odakle mogu da se crpe informacije. Pri kreiranju anotacijske datoteke, biće korišćeno trinaest baza, gde će se ključna informacija o povezanosti gen-bolest izdvajati iz sedam baza koje u nastavku nazivamo izvornim bazama. Preostalih šest baza biće korišćene za dobijanje dodatnih informacija o genima i bolestima i njih u nastavku nazivamo pomoćnim bazama.

1.1 Opis problema

Kreiranje anotacijske datoteke podrazumeva objedinjavanje postojećih informacija o vezama gen-bolest iz različitih izvornih baza podataka. Pored informacije o vezi gen-bolest, koja se dobija iz izvornih baza, anotacijska datoteka sadrži i dodatne informacije o konkretnom genu i konkretnoj bolesti koje se mogu preuzeti iz izvornih ili iz pomoćnih baza podataka. Informacije o povezanosti bolesti sa odgovarajućim genima mogu se dobiti na dva načina:

- pretragom specijalizovanih baza podataka
- metodama istraživanja teksta

Uopšteno, proces kreiranja anotacijske datoteke može se opisati u dva koraka:

1. **Formiranje anotacijske datoteke** - u ovom koraku prikupljaju se veze gen-bolest iz izvornih baza i popunjavaju se kolone vezane za gen, bolest i izvor veze.
2. **Pretraga dodatnih informacija o genima i bolestima** - u ovom koraku na osnovu poznatih vrednosti veze gen-bolest vrši se pretraga dodatnih informacija o genima i bolestima na osnovu informacija sadržanih unutar izvornih i pomoćnih baza. Nekada se ove informacije (ili neke od njih) mogu pronaći u istoj bazi odakle je preuzeta veza gen-bolest ali to najčešće nije slučaj i neophodno je pretražiti i druge baze.

Kreiranje anotacijske datoteke zahteva implementaciju softvera koji bi služio za prikaz anotacijske datoteke i upravljanje samim parsiranjem (eng. *parsing*) tj. procesom dobijanja anotacijske datoteke. Prilikom kreiranja anotacijske datoteke biće korišćene sledeće specijalizovane baze podataka:

1. Izvorne baze
   - `DisGeNet`, `Cosmic`, `HumsaVar`, `Orphanet`, `ClinVar`, `HPO`, `Diseases`
2. Pomoćne baze
   - `UniProt`, `HUGO`, `OBO`, `RGD`, `Orphanet`, `Ensembl`

Anotacijska datoteka biće zasnovana na sadržaju izvornih baza, dok će se pomoćne baze koristiti kao dodatni resurs ukoliko dodatne informacije o genima i bolestima nije moguće pronaći unutar izvornih baza.

1.2 Istraživanje teksta

Prilikom kreiranja anotacijske datoteke biće korišćene tehnike istraživanja teksta koje podrazumevaju izvođenje visokokvalitetnih informacija iz teksta. To uključuje automatsko otkrivanje novih, ranije nepoznatih informacija iz različitih pisanih izvora [8]. Pisani resursi mogu uključivati veb stranice, knjige, e-poruke, recenzije i članke.

Mogu se razlikovati tri različite perspektive istraživanja teksta: ekstrakcija informacija, istraživanje podataka i otkrivanje znanja u bazama podataka (eng. *KDD - Knowledge Discovery in Database*) [10]. Istraživanje teksta obično uključuje proces strukturiranja ulaznog teksta (parsiranje uz dodavanje nekih izvedenih jezičkih karakteristika i uklanjanje drugih, te naknadno umetanje u bazu podataka), izvođenje obrazaca unutar strukturiranih podataka i konačno evaluaciju i interpretaciju izlaza. Tipični zadaci istraživanja teksta uključuju kategorizaciju teksta, grupisanje teksta, izdvajanje koncepta/entiteta, proizvodnju granularnih taksonomija, analizu sentimenta, sažimanje dokumenta i modeliranje odnosa entiteta (tj. učenje odnosa između imenovanih entiteta).

Istraživanje teksta skoro svakodnevno u svom radu koriste naučnici za istraživanje podataka (eng. *data scientists*) i drugi korisnici za obradu velikih podataka i razvoj algoritama dubokog učenja (eng. *deep learning*) koji mogu analizirati ogromne skupove nestrukturiranih podataka. Istraživanje i analiza teksta pomaže organizacijama da pronađu potencijalno vredne poslovne propuste ili dostignuća uvidom u imejlove klijenata, evidencije poziva, objave na društvenim mrežama, medicinske kartone i druge izvore tekstualnih podataka.

1.3 Struktura anotacijske datoteke

Anotacijska datoteka sadrži osam kolona koje nose informaciju o vezi gen-bolest. Informacije u nekim kolonama se odnose na gen, nekoliko kolona predstavlja samu bolest, dok preostale kolone predstavljaju dodatne informacije koje korisnicima daju dodatna pojašnjenja veze gen-bolest.

Kolone u anotacijskoj datoteci su sledeće:

1. `Gene Symbol` - predstavlja kratki identifikator za određeni gen. Pravila o nomenklaturi gena je uspostavio nomenklaturni komitet organizacije za ljudski genom (eng. *HUGO - Human Genome Organisation*) i kasnije su ih usvojile

druge istraživačke zajednice. Prema pravilima HUGO nomenklature, svaki ljudski gen ima svoj identifikator u punom i u skraćenom obliku. Zahtevi koje Gene Symbol mora da ispuni su: mora da bude jedinstven, da sadrži samo slova engleske abecede i arapske brojeve, ne sme da sadrži znakove interpunkcije i ne sme da sadrži nikakve reference na vrste kojima pripada (npr. H/h za ljude) [18].

2. Entrez ID - jedinstveni celobrojni indentifikator preuzet iz baze Entrez Gene[1].

3. UniProt ID - predstavlja identifikator proteinske sekvence.

4. Ensembl ID - identifikator gena prema bazi Ensembl. (eng. *European Bioinformatics Institute and the Wellcome Trust Sanger Institute*), počinje sa ENS za Ensembl, a zatim sa G za gen nakon čega sledi niz brojeva (npr. ENSG00000139618 je Ensembl ID za BRCA2 gen). Ovaj identifikator je jedinstven i po pravilu nepromenljiv, čak i ako se gen ažurira. U slučaju drugih vrsta osim ljudske, u identifikator se ubacuje kod od tri slova (npr. ENSMUSG0000041147 bi ukazivao na BRCA2 gen kod miša) [11].

5. DOID - ontološki identifikator bolesti (eng. *DOID - Disease Ontology ID*) sastoji se od prefiksa DOID iza kojeg sledi broj (npr. Alzheimer's disease, DOID:10652) [19].

6. Source - predstavlja naziv specijalizovane baze iz koje je preuzeta informacija o vezi gena i bolesti. U slučaju da jedna veza ima više izvora, onda se za svaki izvor veze gen-bolest stavlja u poseban red.

7. Disease Name - sadrži naziv bolesti ili grupe oboljenja.

8. DOID Source - sadrži informaciju na koji način je dobijen DO identifikator (DOID). Pored samog izvora, ova kolona sadrži i meru pouzdanosti pronađenog DOID-a izraženu u procentima. Proces nalaženja vrednosti atributa DOID predstavlja značajan izazov zbog čega je i uvedena ova kolona koja bliže opisuje način pronalaska vrednosti DOID. DOID Source može da sadrži sledeće vrednosti:

[1] Entrez Gene http://www.ncbi.nlm.nih.gov/gene je baza podataka Nacionalnog centra za biotehnološke informacije (eng. *NCBI - National Center for Biotechnology Information*) koja sadrži informacije specifične za gene.

- `Xref ->` { `GARD` | `ICD-10` | `MeSH` | `MedDRA` | `OMIM` | `UMLS` } - označava da je `DOID` pronađen preko nekog od `Xref`-ova koji predstavljaju unakrsne reference koje opisuju analogni termin u nekoj drugoj bazi. Koristi se kao veza ka atributima `DOID` i `Disease Name`. Smatra se da je ovako pronađeni `DOID` 100% tačan.
- `DiseaseName -> Frozenset` - označava da je `DOID` pronađen korišćenjem rečnika[2] i smatra se da je `DOID` 100% tačan.
- `Database` - označava da je `DOID` sastavni deo izvorne baze iz koje je nastao red u anotacijskoj datoteci, smatra se da je `DOID` 100% tačan.
- `DiseaseName -> Typesense,` x`%` - označava da je `DOID` pronađen korišćenjem tehnike punog pretraživanja teksta (eng. *full text search*) tj. korišćenjem pretraživača pod imenom `Typesense`[3] o čemu će biti više reči u poglavlju 4.2; x predstavlja broj koji je izražen u procentima i označava koliko je bolest slična sa bolešću za koju je vezan pronađeni `DOID`.

U tabeli 1.1 prikazan je sažeti opis kolona anotacijske datoteke, dok je deo anotacijske datoteke prikazan u tabeli 1.2.

[2]U nastavku će detaljnije biti opisan princip pronalaženja `DOID`-a preko rečnika.

[3]`Typesense` je besplatan pretraživač koji je tolerantan na sitne greške u upitima `https://typesense.org/`.

	Vrsta veze	Baze gde se može preuzeti	Primer
Gene Symbol	Gen	DisGeNet, COSMIC, ClinVar, HumsaVar, Orphanet, HPO, Diseases, UniProt, HUGO	ECE1
Entrez ID	Gen	ClinVar, COSMIC, DisGeNet, HPO, UniProt, HUGO, Ensembl	1889
Uniprot ID	Gen	HumsaVar, Orphanet, UniProt, HUGO, Ensembl	P42892
Ensembl ID	Gen	Orphanet, UniProt, HUGO, Ensembl	ENSG00000117298
DOID	Bolest	Diseases, OBO, RGD	DOID:10825
Source	Gen-Bolest	DisGeNet, COSMIC, ClinVar, HPO, HumsaVar, Orphanet, Diseases	DisGeNet
Disease Name	Bolest	ClinVar, COSMIC, Diseases, DisGeNet, HumsaVar, Orphanet, Orphanet Xref, RGD, OBO	Yao syndrome
DOID Source	Bolest	/	Xref -> OMIM

Tabela 1.1: Sažeti opis kolona anotacijske datoteke

Gene Symbol	Entrez ID	Uniprot ID	Ensembl ID	DOID	Source	Disease Name	DOID Source
A1BG	1	P04217	ENSG00000121410.12	DOID:5419	DisGeNet	Schizophrenia	Xref -> UMLS
A1BG	1	P04217	ENSG00000121410.12	DOID:9005369	DisGeNet	Hepatomegaly	DiseaseName -> Frozenset
IL6	3569	P05231	ENSG00000136244	DOID:9279	Diseases	Hyperhomocysteinemia	Database
RPS19	6223	P39019	ENSG00000116218	DOID:9279	Diseases	Hyperhomocysteinemia	Database
PAX3	5077	P23760	ENSG00000135903.20	DOID:4051	ClinVar	Alveolar rhabdomyosarcoma	Xref -> UMLS
PAX3	5077	P23760	ENSG00000135903.20	DOID:4051	Cosmic	alveolar rhabdomyosarcoma	DiseaseName -> Frozenset
PAX3	5077	P23760	ENSG00000135903.20	DOID:4051	DisGeNet	Alveolar rhabdomyosarcoma	Xref -> UMLS
PAX3	5077	P23760	ENSG00000135903.20	DOID:4051	HPO	alveolar rhabdomyosarcoma	Xref -> OMIM
IL36RN	26525	Q9UBH0	ENSG00000136695	DOID:8503	Orphanet	Generalized pustular psoriasis	Xref -> ICD-10
NOD2	64127	Q9HC29	ENSG00000167207	DOID:9004527	HumsaVar	Yao syndrome	DiseaseName -> Frozenset
AQP2	359	P41181	ENSG00000167580	DOID:557	Diseases	Kidney disease	Database
CTSK	1513	P43235	ENSG00000143387	DOID:9000472	DisGeNet	Disproportionate short stature	DiseaseName -> Typesense, 38%

Tabela 1.2: Deo anotacijske datoteke

Glava 2

Podaci

U ovom poglavlju biće opisani podaci koji se koriste prilikom procesa kreiranja anotacijske datoteke. Podaci predstavljaju najbitniji deo ovog procesa. Prikupljani su godinama od strane različitih organizacija i predstavljeni kao specijalizovane baze podataka u različitim formatima. Ključni kriterijum za odabir specijalizovane baze podataka u ovom radu je njena ažurnost. Baza koja nije ažurirana više od godinu dana ne uzima se u obzir prilikom odabira. Drugi važan kriterijum je licenca pod kojom se podaci u bazi podataka nalaze. Uzimaju se u obzir samo baze podataka sa otvorenom licencom ili baze koje dozvoljavaju korišćenje podataka u nekomercijalne svrhe. Većina baza sadrži relevatne informacije o povezanosti gena i bolesti, koje su formatirane kao `.tsv`, `.csv`, `.obo`, `.dat`, `.xml` ili `.txt` i na jednostavan način se mogu preuzimati.

Datoteke koje potiču iz različitih izvora mogu u velikoj meri varirati količinom informacija koje nude korisniku. Ove informacije možemo podeliti na dve grupe:

1. esencijalne ili ključne
2. pomoćne

Esencijalne informacije su one koje se direktno odnose na gen i odgovarajuću bolest prouzrokovanu promenama na datom genu. Pomoćne informacije mogu da definišu tačan položaj gena u genomu, izvor odakle je veza gen-bolest preuzeta, tip tkiva koji bolest pogađa, identifikacija `Xref`, identifikacija `OMIM` i tome slično. Neke od ovih informacija kao što je izvor su bitne za predmet istraživanja, dok je recimo tačan položaj gena u genomu podatak koji može da se zanemari u ovom istraživanju.

Kao što je već napomenuto, u izradi anotacijske datoteke biće korišćeno trinaest specijalizovanih baza podataka koje će biti podeljene na dve grupe, izvorne i pomoćne baze.

2.1 Izvorne baze podataka

Izvorne baze podataka predstavljaju osnovu za anotacijsku datoteku. Parsiranjem izvornih baza zadržavamo nama bitne informacije koje treba sačuvati unutar anotacijske datoteke. Izvorne baze koriste se kao osnova tj. slogovi koji se nalaze unutar izvornih baza formiraju anotacijsku datoteku, gde se nedostajuće kolone tih slogova dalje dopunjuju relevantnim informacijama iz drugih baza. Celokupan proces parsiranja će biti detaljnije opisan u poglavlju 4. Za kreiranje anotacijske datoteke korišćeno je sedam izvornih baza: `DisGeNet`, `Cosmic`, `HumsaVar`, `Orphanet`, `ClinVar`, `HPO` i `Diseases`, koje će detaljnije biti opisane u nastavku ovog poglavlja. Atributi izvornih baza koriste se kao atributi anotacijske datoteke ili služe kao veza za njihovo pronalaženje.

2.1.1 DisGeNet

`DisGeNet` predstavlja platformu koja sadrži jednu od najvećih javno dostupnih kolekcija gena i njihovih varijanti povezanih sa ljudskim bolestima. Podaci su homogeno obeleženi kontrolisanim rečnicima[1] (eng. *Controlled Vocabularies*). Takođe, `DisGeNet` nudi skup bioinformatičkih alata za analizu ovih podataka. Samu bazu održava grupa za integrativnu biomedicinsku informatiku (eng. *IBI - Integrative Biomedical Informatics*) sa sedištem u Barseloni, Španija [15]. Baza `DisGeNet` ima mnoštvo atributa od kojih će pri kreiranju anotacijske datoteke biti korišćeni `geneId`, `geneSymbol`, `diseaseId` i `diseaseName` koji su prikazani u tabeli 2.1. U narednoj listi biće dat opis atributa baze `DisGeNet` kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `geneSymbol` - predstavlja simbol gena i koristi se kao `Gene Symbol` atribut anotacijske datoteke.

[1]Kontrolisani rečnici su standardizovane i organizovane liste unapred definisanih termina i fraza, koje se koriste prilikom indeksiranja i pronalaženja željenih informacija. Kontrolisani rečnici osiguravaju da će isti podaci biti opisani korišćenjem istog željenog termina svaki put kada se indeksira i to će olakšati pronalaženje svih informacija o određenom podatku tokom procesa pretraživanja.

2. `geneId` - predstavlja identifikator gena prema bazi podataka `Entrez Gene` i koristi se kao `Entrez ID` atribut anotacijske datoteke.

3. `diseaseName` - predstavlja ime bolesti i koristi se kao `Disease Name` atribut anotacijske datoteke.

4. `diseaseId` - predstavlja vrednost `Xref` koja se naziva `UMLS_CUI`[2]. Vrednosti `Xref` služe kao veze prema atributima `DOID` i `Disease Name`, gde se sam atribut `diseaseId` koristi kao veza za pronalazak atributa `DOID` anotacijske datoteke.

geneId	**geneSymbol**	**...**	**diseaseId**	**diseaseName**	**diseaseType**	**...**
1	A1BG	...	C0019209	Hepatomegaly	phenotype	...
1	A1BG	...	C0036341	Schizophrenia	disease	...
2	A2M	...	C0002395	Alzheimer's Disease	disease	...
2	A2M	...	C0007102	Malignant tumor of colon	disease	...
2	A2M	...	C0009375	Colonic Neoplasms	group	...

Tabela 2.1: Deo baze `DisGeNet`

Datoteka koja se koristi prilikom parsiranja baze `DisGeNet` može se pronaći pod nazivom `curated_gene_disease_associations.tsv` i besplatno preuzeti sa veb stranice putem sledećeg linka `https://www.disgenet.org/` (poslednji put pristupano avgust 2022).

2.1.2 COSMIC

`COSMIC` (eng. *the Catalogue of Somatic Mutations in Cancer*) je najveći globalni izvor informacija o somatskim mutacijama u vezi sa ljudskim kancerom. `COSMIC` se sastoji od baze podataka `COSMIC` i projekta `Cell Lines`, dva odvojena, ali povezana resursa. Baza `COSMIC` sastoji se od podataka visoke preciznosti, ručno sakupljanih od strane stručnjaka u preko 27000 recenziranih radova. Fokus ovih radova su geni za

[2]`UMLS` (eng. *Unified Medical Language System*) predstavlja jedinstveni identifikator biomedicinskog rečnika koji je kontrolisan od strane `NLM` (eng. *National Library of Medicine*).

koje je poznato da su povezani sa kancerom ili za koje se sumnja, kao i na njihovim mutacijama [6]. Deo baze COSMIC prikazan je u tabeli 2.2. U narednoj listi biće dat opis atributa baze COSMIC kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. Gene Symbol - predstavlja simbol gena i koristi se kao Gene Symbol atribut anotacijske datoteke.

2. Entrez GeneId - predstavlja identifikator gena prema bazi podataka Entrez Gene i koristi se kao Entrez ID atribut anotacijske datoteke.

3. Tumour Types(Somatic) i Tumour Types(Germline) - predstavljaju ime bolesti i koriste se kao Disease Name atribut anotacijske datoteke, s tim što ukoliko obe kolone sadrže vrednost, svaka bolest će biti navedena u posebnom redu u anotaijskoj datoteci.

Baza COSMIC pruža nekoliko datoteka za slobodno preuzimanje uz neophodnu registraciju sa akademske mreže. Datoteka koja se koristi prilikom parsiranja baze COSMIC može se pronaći pod nazivom cancer_gene_census.csv i besplatno preuzeti sa veb stranice putem sledećeg linka https://cancer.sanger.ac.uk/cosmic/download (poslednji put pristupano avgust 2022). Postoje tri verzije ove datoteke: whole, filtered i scripted. Za potrebe kreiranja anotacijske datoteke korišćena je verzija whole.

2.1.3 HumsaVar

Baza HumsaVar je deo veće baze UniProt (Swiss-Prot Protein Knowledgebase), sastoji se od ljudskih genskih varijanti sakupljanih od strane stručnjaka iz različitih naučnih izveštaja [17]. Ova baza sadrži informacije o patogenosti više od 70 000 ljudskih varijanti. Većina varijanti je označena ili kao neutralni polimorfizam ili kao varijanta povezana sa bolešću (39.5% i 50.0%, respektivno), sa malim brojem neklasifikovanih unosa (10.6%). Iako datoteka HumsaVar uključuje pristup UniProt-u, ona se ne povezuje direktno sa eksperimentalnim strukturama niti pruža ukupan uvid u odnos između strukture i bolesti [27]. Baza HumsaVar sadrži nekoliko korisnih atributa za kreiranje anotacijske datoteke kao što su Main gene name, Swiss-Prot AC, Disease name i Variant category koji su prikazani u tabeli 2.3. U narednoj listi biće dat opis atributa baze HumsaVar kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `Main gene name` - predstavlja simbol gena i koristi se kao `Gene Symbol` atribut anotacijske datoteke.

2. `Swiss-Prot AC` - predstavlja identifikator proteinske sekvence i koristi se kao `UniProt ID` atribut anotacijske datoteke.

3. `Variant category` - predstavlja različite kategorije bolesti i može sadržati sledeće vrednosti: `LP/P` (eng. *likely pathogenic or pathogenic*), `LB/B` (eng. *likely benign or benign*) i `US` (eng. *uncertain significance*).

4. `Disease name` - predstavlja ime bolesti i koristi se kao `Disease Name` atribut anotacijske datoteke, samo za one redove gde je vrednost atributa `Variant category` jednako `LP/P`.

Datoteka koja se koristi prilikom parsiranja baze `HumsaVar` može se pronaći pod nazivom `humsavar.txt` i besplatno preuzeti sa veb stranice putem sledećeg linka `https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/complete/docs/humsavar` (poslednji put pristupano avgust 2022).

Gene Symbol	Name	Entrez GeneId	...	Tumour Types(Somatic)	Tumour Types(Germline)	...
AR	Androgen Receptor	367	...	prostate	/	...
FH	fumarate hydratase	2271	...	/	leiomyomatosis, renal	...
ALK	anaplastic lymphoma kinase (Ki-1)	238	...	ALCL, NSCLC, neuroblastoma, inflammatory myofibroblastic tumour, Spitzoid tumour	neuroblastoma	...
APC	adenomatous polyposis of the colon gene	324	...	colorectal, pancreatic, desmoid, hepatoblastoma, glioma, other CNS	colorectal, pancreatic, desmoid, hepatoblastoma, glioma, other CNS	...
ATM	ataxia telangiectasia mutated	472	...	T-PLL	leukaemia, lymphoma, medulloblastoma, glioma	...

Tabela 2.2: Deo baze `COSMIC`

Main gene name	Swiss-Prot AC	FTId	AA change	Variant category	dbSNP	Disease name
ABCA1	O95477	VAR_009146	p.Arg587Trp	LP/P	rs2853574	Tangier disease (TGD) [MIM:205400]
ABCA3	Q99758	VAR_035728	p.Leu290Met	US	-	A breast cancer sample
ABCA2	Q9BZC7	VAR_044526	p.Pro583His	LB/B	rs908828	-
ABCA4	P78363	VAR_008438	p.Glu1122Lys	LP/P	rs61751399	Stargardt disease 1 (STGD1) [MIM:248200]
ABCB11	O95342	VAR_013339	p.Arg1268Gln	LP/P	rs72549394	Cholestasis, progressive familial intrahepatic, 2 (PFIC2) [MIM:601847]

Tabela 2.3: Deo baze `HumsaVar`

2.1.4 Orphanet

`Orphanet` je jedinstveni resurs koji prikuplja informacije i unapređuje znanje o retkim bolestima kako bi se poboljšala dijagnostika, nega i lečenje pacijenata sa retkim bolestima. `Orphanet` ima za cilj da pruži visokokvalitetne informacije o retkim bolestima i obezbedi jednak pristup znanju za sve zainteresovane strane. `Orphanet` takođe održava nomenklaturu retkih bolesti (`ORPHAcode`), od suštinskog značaja za poboljšanje vidljivosti retkih bolesti u zdravstvenim i istraživačkim informacionim sistemima. Osnovan je u Francuskoj od strane `INSERMa` (eng. *French National Institute for Health and Medical Research*) 1997. godine, i postepeno je prerastao u konzorcijum od 40 zemalja, unutar Evrope i širom sveta.

```
<Disorder id="5">
  <OrphaCode>93</OrphaCode>
   ...
  <Name lang="en">Aspartylglucosaminuria</Name>
  <DisorderType id="21394">
    <Name lang="en">Disease</Name>
  </DisorderType>
  <DisorderGeneAssociationList count="1">
       ...
      <Gene id="15470">
         ...
        <Symbol>AGA</Symbol>
         ...
        <ExternalReferenceList count="6">
           ...
          <ExternalReference id="56681">
            <Source>Ensembl</Source>
            <Reference>ENSG00000038002</Reference>
          </ExternalReference>
           ...
          <ExternalReference id="32441">
            <Source>SwissProt</Source>
            <Reference>P20933</Reference>
          </ExternalReference>
```

```
            ...
          </ExternalReferenceList>
        </Gene>
      ...
    </DisorderGeneAssociationList>
  </Disorder>
```

Listing 2.1: Deo baze `Orphanet`

Baza `Orphanet` ima mnoštvo atributa od kojih će pri kreiranju anotacijske datoteke biti korišćeni `Symbol`, `Ensembl`, `Name` i `OrphaCode` koji su prikazani u listingu 2.1. U narednoj listi biće dat opis atributa baze `Orphanet` kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `Symbol` - predstavlja simbol gena i koristi se kao `Gene Symbol` atribut anotacijske datoteke.

2. `Ensembl` - predstavlja identifikator gena prema bazi podataka `Ensembl` i koristi se kao `Ensembl ID` atribut anotacijske datoteke.

3. `Name` - predstavlja ime bolesti i koristi se kao `Disease Name` atribut anotacijske datoteke.

4. `OrphaCode` - predstavlja jedinstveni numerički identifikator koji je nasumično dodeljen od strane organizacije `Orphanet` prilikom stvaranja entiteta. Trenutno, `ORPHACode` se sastoji od jedne cifre ili najviše šest cifara, dok se u budućnosti broj cifara može proširiti. `ORPHACode` se koristi kao veza prema `Xref` vrednostima iz druge datoteke baze `Orphanet`, gde pronađene `Xref` vrednosti dalje služe kao veza za pronalaženje atributa `DOID` anotacijske datoteke.

Postoji mnoštvo različitih datoteka u okviru baze `Orphanet`. Datoteka koja se koristi prilikom parsiranja baze `Orphanet` može se pronaći pod nazivom `en_product6.xml` i besplatno preuzeti sa veb stranice putem sledećeg linka `http://www.orphadata.com/data/xml/en_product6.xml` (poslednji put pristupano avgust 2022).

2.1.5 ClinVar

ClinVar je besplatno dostupna, javna arhiva ljudskih genetskih varijanti i tumačenja njihovih odnosa sa bolestima i drugim stanjima, koju održava NIH (eng. *National Institutes of Health*). Dostavljena tumačenja varijanti se objedinjuju i stavljaju na raspolaganje na veb lokaciji baze ClinVar (https://www.ncbi.nlm.nih.gov/clinvar/), i kao datoteke za preuzimanje pomoću FTPa i preko programskih alata kao što su NCBI E-utilities [13].

Prilikom kreiranje anotacijske datoteke korišćeni su sledeći atributi baze ClinVar: #GeneID, AssociatedGenes, RelatedGenes, ConceptID, DiseaseName i DiseaseMIM koji su prikazani u tabeli 2.4. U narednoj listi biće dat opis atributa baze ClinVar kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. AssociatedGenes i RelatedGenes - predstavljaju simbol gena i koriste se kao Gene Symbol atribut anotacijske datoteke, ukoliko obe kolone sadrže vrednost, svaki simbol gena će biti naveden u posebnom redu u anotacijskoj datoteci.

2. #GeneID - predstavlja identifikator gena prema bazi podataka Entrez Gene i koristi se kao Entrez ID atribut anotacijske datoteke.

3. DiseaseName - predstavlja ime bolesti i koristi se kao Disease Name atribut anotacijske datoteke.

4. ConceptID i DiseaseMIM - predstavljaju Xref vrednosti UMLS_CUI i OMIM respektivno i koriste se kao veza za pronalazak atributa DOID anotacijske datoteke.

Datoteka koja se koristi prilikom parsiranja baze ClinVar može se pronaći pod nazivom gene_condition_source_id.txt i besplatno preuzeti sa veb stranice putem sledećeg linka https://ftp.ncbi.nlm.nih.gov/pub/clinvar/gene_condition_source_id (poslednji put pristupano avgust 2022).

#GeneID	AssociatedGenes	RelatedGenes	ConceptID	DiseaseName	...	DiseaseMIM	...
351	APP	/	C0002395	Alzheimer disease	...	/	...
324	APC	/	C0346629	Colorectal cancer	...	114500	...
200894	ARL13B	/	C2676771	Joubert syndrome 8	...	612291	...
286410	ATP11C	/	C4746970	X-linked congenital hemolytic anemia	...	301015	...
6347	CCL2	/	C0027794	Neural tube defect	...	182940	...

Tabela 2.4: Deo baze `ClinVar`

...	symbol	...	alias_symbol	...	entrez_id	ensembl_gene_id	...	uniprot_ids	...
...	ABHD15	...	/	...	116236	ENSG00000168792	...	Q6UXT9	...
...	ACAD9-DT	...	/	...	123706524	ENSG00000287110	...	/	...
...	ACP4	...	/	...	93650	ENSG00000142513	...	Q9BZG2	...
...	ACY3	...	HCBP1|MGC9740|ACY-3	...	91703	ENSG00000132744	...	Q96HD9	...
...	ADAM2	...	PH-30b|PH30|CT15	...	2515	ENSG00000104755	...	Q99965	...

Tabela 2.5: Deo baze `HUGO`

2.1.6 HPO

Projekat HPO (eng. *The Human Phenotype Ontology*) pruža ontologiju medicinski relevantnih fenotipova, odnosa bolest-fenotip i algoritme koji rade sa njima. HPO se trenutno razvija koristeći medicinsku literaturu, Orphanet, DECIPHER i OMIM. Ova baza trenutno sadrži preko 13000 termina i preko 156000 napomena o naslednim bolestima.

entrez-gene-id	**entrez-gene-symbol**	...	**G-D source**	**disease-ID for link**
8192	CLPP	...	mim2gene	OMIM:614129
90121	TSR2	...	orphadata	ORPHA:124
8200	GDF5	...	mim2gene	OMIM:610017
8200	GDF5	...	orphadata	ORPHA:968
8200	GDF5	...	mim2gene	OMIM:615072

Tabela 2.6: Deo baze HPO

Doprinos baze HPO prilikom kreiranja anotacijske datoteke nije prevelik posmatrano iz ugla lako dostupnih informacija, koje se odmah mogu preuzeti i sačuvati unutar anotacijske datoteke. Ipak, HPO sadrži nekoliko korisnih atributa za kreiranje anotacijske datoteke kao što su `entrez-gene-symbol`, `entrez-gene-id` i `disease-ID for link` koji su prikazani u tabeli 2.6. U narednoj listi biće dat opis atributa baze HPO kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `entrez-gene-symbol` - predstavlja simbol gena i koristi se kao `Gene Symbol` atribut anotacijske datoteke.

2. `entrez-gene-id` - predstavlja identifikator gena prema bazi podataka `Entrez Gene` i koristi se kao `Entrez ID` atribut anotacijske datoteke.

3. `disease-ID for link` - predstavlja vezu prema imenu bolesti u vidu kodova ORPHA i OMIM. Kod ORPHA predstavlja sigurniju vezu iz razloga što pronalaženje imena bolesti na osnovu koda ORPHA daje jedinstveno ime bolesti, dok kod koda OMIM to nije slučaj. Ukoliko preko koda OMIM postoji više od jednog imena

bolesti, svaka bolest će biti navedena u posebnom redu u anotacijskoj datoteci. Pronađeno ime bolesti koristi se kao `Disease Name` atribut anotacijske datoteke.

Datoteka koja se koristi prilikom parsiranja baze `HPO` može se pronaći pod nazivom `genes_to_phenotypes.txt` i besplatno preuzeti sa veb stranice putem sledećeg linka `http://purl.obolibrary.org/obo/hp/hpoa/genes_to_phenotype.txt` (poslednji put pristupano avgust 2022).

2.1.7 Diseases

Baza `Diseases` ima za cilj da bude najsveobuhvatnija besplatna, javno dostupna baza podataka o vezama bolesti i gena. `Diseases` integriše podatke o povezanosti gena i bolesti primenom metoda istraživanja teksta na informacije koje se ručno izdvajaju iz različitih naučnih radova. Svakom unosu dodeljuje se ocena pouzdanosti koja olakšava poređenje unosa različitih vrsta iz različitih izvora. Postoji veb aplikacija[3] otvorenog koda koja prepoznaje bolesti i ljudske gene u tekstu i izdvaja veze bolesti i gena. Baza `Diseases` crpi informacije iz sledećih baza: `GHR` (eng. *Genetics Home Reference*), `UniProtKB` (eng. *UniProt Knowledgebase*), rezultate genomskih veza iz `DistiLD` (eng. *diseases and traits in linkage disequilibrium blocks*) i podatke o mutacijama iz `COSMIC` (eng. *Catalog of Somatic Mutations in Cancer*) [16].

Baza `Diseases` predstavlja jednu od bitnijih izvornih baza prilikom kreiranja anotacijske datoteke iz razloga što sadrži u celosti informaciju o bolesti, koja je predstavljena atributima `DOID` i `Disease Name` anotacijske datoteke. Baza `Diseases` sadrži nekoliko korisnih atributa za kreiranje anotacijske datoteke kao što su `Gene Identifier`, `Gene Name`, `Disease Identifier` i `Disease Name` koji su prikazani u tabeli 2.7. U narednoj listi biće dat opis atributa baze `Diseases` kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `Gene Name` - predstavlja simbol gena i koristi se kao `Gene Symbol` atribut anotacijske datoteke.

2. `Gene Identifier` - predstavlja identifikator proteina i koristi se kao veza prema atributima `Ensembl ID`, `UniProt ID` i `Entrez ID` anotacijske datoteke.

[3]Aplikacija se može pronaći putem sledećeg linka: `https://diseases.jensenlab.org/Search`.

3. `Disease Identifier` - predstavlja identifikator bolesti i koristi se kao `DOID` atribut anotacijske datoteke.

4. `Disease Name` - predstavlja ime bolesti i koristi se kao `Disease Name` atribut anotacijske datoteke.

Gene Identifier	**Gene Name**	**Disease Identifier**	**Disease Name**	...
18S_rRNA	18S_rRNA	DOID:162	Cancer	...
ENSP00000387699	CREB1	DOID:8670	Eating disorder	...
ENSP00000387662	GCG	DOID:6977	Pancreatic cholera	...
ENSP00000388620	EOMES	DOID:417	Autoimmune disease	...
ENSP00000387760	ENSP00000387760	DOID:1406	Iritis	...

Tabela 2.7: Deo baze `Diseases`

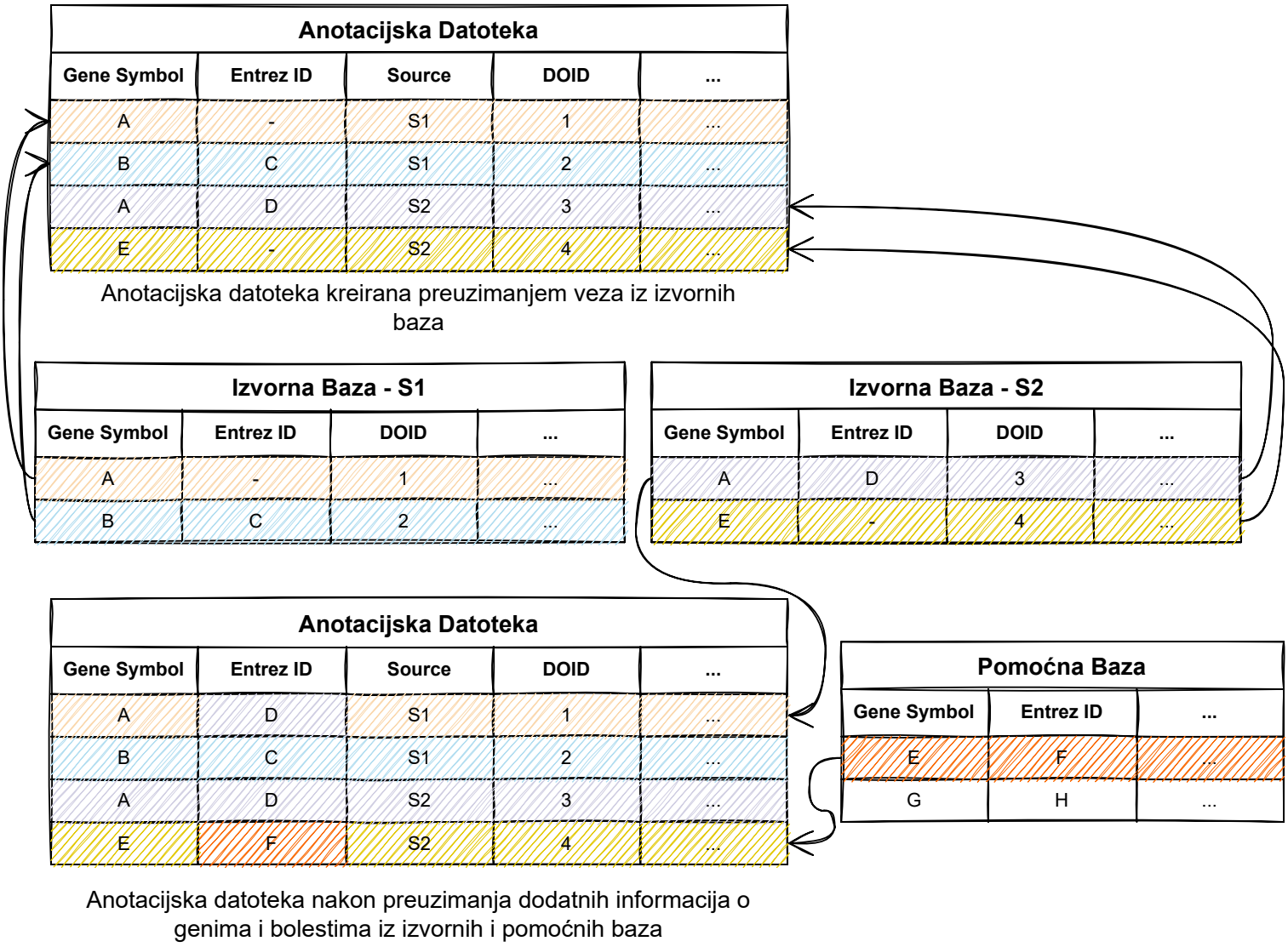
Baza `Diseases` pruža četiri različite grupe datoteka: `Text mining channel`, `Knowledge channel`, `Experiments channel` i `Integrated channel - experimental`, u dva formata `full` i `filtered`. Datoteka koja se koristi prilikom parsiranja baze `Diseases` naziva se `human_disease_textmining_filtered.tsv` i pripada `Text mining channel` grupi u `filtered` formatu. Može se besplatno preuzeti putem sledećeg linka `https://diseases.jensenlab.org/Downloads` (poslednji put pristupano avgust 2022).

2.2 Pomoćne baze podataka

Pomoćne baze predstavljaju dodatni resurs izvornim bazama prilikom kreiranja anotacijske datoteke. Kao što je već pomenuto, anotacijska datoteka kao osnovu sadrži veze gen-bolest preuzete iz izvornih baza. Pored informacije o vezi gen-bolest, anotacijska datoteka sadrži i dodatne informacije o genima i bolestima. Dodatne informacije za svaku vezu gen-bolest se nekada mogu pronaći u izvornoj bazi odakle je i preuzeta informacija o samoj vezi, nekada u nekoj drugoj izvornoj bazi, a nekada iz baza koje nisu ušle u skup izvornih baza podataka, takozvanih pomoćnih baza

podataka. Pomoćne baze podataka ne koriste se za prikupljanje veza gen-bolest već samo za prikupljanje dodatnih podataka o pojedinačnim genima i bolestima.

Nakon što su opisane izvorne i pomoćne baze na primeru će biti objašnjena njihova uloga i biće prikazane njihove razlike.



Slika 2.1: Korišćenje izvornih i pomoćnih baza

Na slici 2.1 dat je prikaz korišćenja izvornih i pomoćnih baza na primeru kreiranja anotacijske datoteke. U ovom primeru koriste se dve izvorne baze `S1` i `S2` koje su zadužene za kreiranje osnove anotacijske datoteke tj. za preuzimanje veza gen-bolest koje će biti sadržane unutar anotacijske datoteke i za preuzimanje dodatnih

informacija o genima i bolestima. Još jedna baza se koristi za preuzimanje dodatnih informacija o genima i bolestima u ovom primeru, a to je pomoćna baza koja se koristi kao dodatni resurs izvornim bazama prilikom ove vrste preuzimanja.

U navedenom primeru izvorne baze daju četiri veze, od kojih dve sadrže sve informacije o vezi gen-bolest, dok dve veze zahtevaju dodatno preuzimanje dodatnih informacija o genima i bolestima. Bez gubljenja opštosti u ovom primeru fokus će biti na preuzimanju dodatnih informacija o genima, bez bolesti. Shodno tome, veza anotacijske datoteke preuzeta iz izvorne baze `S1` čija je vrednost atributa `Gene Symbol` jednaka *A* zahteva pronalaženje vrednosti atributa `Entrez ID`. Isto tako, pronalaženje vrednosti ovog atributa zahteva i veza preuzeta iz izvorne baze `S2` čija je vrednost atributa `Gene Symbol` jednaka *E*. Na slici 2.1 je jasno prikazano da se preuzimanje dodatnih informacija o genu za vezu preuzetu iz izvorne baze `S1` vrši korišćenjem izvorne baze `S2`, dok se ova vrsta preuzimanja za vezu preuzetu iz izvorne baze `S2` vrši korišćenjem pomoćne baze.

Dakle, pomoćne baze ne pružaju veze gen-bolest koje će biti sadržane unutar anotacijske datoteke, ali pružaju dodatne informacije za te veze kao što je prikazano u ovom primeru. Prilikom kreiranja anotacijske datoteke korišćeno je šest pomoćnih baza: `Uniprot`, `HUGO`, `OBO`, `RGD`, `Orphanet Xref` i `Ensembl`, koje će detaljnije biti opisane u nastavku ovog poglavlja.

2.2.1 UniProt

Baza `UniProt` (eng. *The Universal Protein Resource*) predstavlja sveobuhvatan izvor za proteinske sekvence i anotacijske podatke. `UniProt` se sastoji od tri baze `UniProtKB`, `UniRef` (eng. *The UniProt Reference Clusters*) i `UniParc` (eng. *UniProt Archive*). Održavanjem baze `UniProt` bavi se konzorcijum `UniProt` i institucije `EMBL-EBI` (eng. *European Bioinformatics Institute*), `SIB` (eng. *Swiss Institute of Bioinformatics*) i `PIR` (eng. *The Protein Information Resource*), gde učestvuje više od 100 ljudi koji su zaduženi za različite zadatke kao što su prikupljanje podataka, razvoj softvera i podršku [25]. Baza `UniProt` ima mnoštvo atributa od kojih će pri kreiranju anotacijske datoteke biti korišćeni `Gene_Name`, `Ensembl`, `UniProtKB-ID`, `GeneID`, `STRING` i `Gene_Synonym` koji su prikazani u listingu 2.2, gde su zelenom bojom označene vrednosti koje se uzimaju za spomenute atribute. U narednoj listi biće dat opis atributa baze `UniProt` kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `Gene_Name` - predstavlja simbol gena i može se iskoristiti kao `Gene Symbol` atribut anotacijske datoteke.

2. `Ensembl` - predstavlja identifikator gena prema bazi podataka `Ensembl` i može se iskoristiti kao `Ensembl ID` atribut anotacijske datoteke.

3. `GeneID` - predstavlja identifikator gena prema bazi podataka `Entrez Gene` i može se iskoristiti kao `Entrez ID` atribut anotacijske datoteke.

4. `UniProtKB-ID` - predstavlja identifikator proteinske sekvence i može se iskoristiti kao `UniProt ID` atribut anotacijske datoteke.

5. `STRING` - predstavlja identifikator proteina koji se koristi kao veza između izvornih i pomoćnih baza prilikom dohvatanja nedostajućih atributa anotacijske datoteke.

6. `Gene_Synonym` - predstavlja sinonim simbola gena i koristi se kao veza za dohvatanje nedostajućih atributa anotacijske datoteke vezanih za genski deo veze.

Datoteka koja se koristi prilikom parsiranja baze `UniProt` može se pronaći pod nazivom `HUMAN_9606_idmapping.dat` i besplatno preuzeti sa veb stranice putem sledećeg linka `https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/by_organism/HUMAN_9606_idmapping.dat.gz` (poslednji put pristupano avgust 2022).

```
Q66LE6 UniProtKB-ID   2ABD_HUMAN
Q66LE6  Gene_Name  PPP2R2D
Q66LE6  Gene_Synonym   KIAA1541
Q66LE6  STRING 9606.ENSP00000399970
Q66LE6  Ensembl    ENSG00000175470.20
Q66LE6  GeneID 55844
        ...
```

Listing 2.2: Deo baze `UniProt`

2.2.2 HUGO

HGNC (eng. *HUGO Gene Nomenclature Committee*) predstavlja neprofitno telo koje zajednički finansiraju američki institut NHGRI (eng. *National Human Genome Research Institute*) i britanski institut Wellcome. Radi pod okriljem organizacije HUGO (eng. *Human Genome Organisation*), uz ključne savete o politici same organizacije od strane SAB (eng. *Scientific Advisory Board*).

HGNC ili drugačije HUGO predstavlja bazu koja odobrava jedinstvene simbole i imena za ljudske lokuse[4] (eng. *human loci*), uključujući gene koji kodiraju proteine (eng. *protein-coding genes*), ncRNA (eng. *non-coding RNA*) gene i pseudogene[5], kako bi se omogućila nedvosmislena naučna komunikacija. Za svaki poznati ljudski gen odobrava se naziv gena i simbol. Svaki simbol je jedinstven i garantovano je da svaki gen dobije samo jedan odobreni simbol. Odobreno je skoro 43 000 simbola, oko 19 000 njih je za gene koji kodiraju proteine, a ostatak uključuje pseudogene, nekodirajuće RNK i genomske karakteristike. Simbole koriste naučnici, naučni časopisi (npr. *Nature Genetics, Genome Research*) i baze podataka kao što su Ensembl, NCBI Gene, MGI, RGD i OMIM [5]. Na bazu HUGO može se gledati kao na pomoćnu bazu koja predstavlja glavni izvor nedostajućih vrednosti atributa anotacijske datoteke vezanih za genski deo veze. Baza HUGO ima mnoštvo atributa od kojih će pri kreiranju anotacijske datoteke biti korišćeni symbol, entrez_id, ensembl_gene_id, uniprot_ids i alias_symbol koji su prikazani u tabeli 2.5. U narednoj listi biće dat opis atributa baze HUGO kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. symbol - predstavlja simbol gena i može se iskoristiti kao Gene Symbol atribut anotacijske datoteke.

2. ensembl_gene_id - predstavlja identifikator gena prema bazi podataka Ensembl i može se iskoristiti kao Ensembl ID atribut anotacijske datoteke.

3. entrez_id - predstavlja identifikator gena prema bazi podataka Entrez Gene i može se iskoristiti kao Entrez ID atribut anotacijske datoteke.

4. uniprot_ids - predstavlja listu identifikatora proteinske sekvence i može se iskoristiti kao UniProt ID atribut anotacijske datoteke.

[4]Lokus je u genetici i evolucijskim proračunima specifična lokacija gena ili DNK sekvence na hromozomu.

[5]Pseudogeni su nefunkcionalni srodnici gena koji su izgubili sposobnost kodiranja proteina ili više ne bivaju eksprimirani u ćeliji [4, 26].

5. `alias_symbol` - predstavlja listu sinonima za simbol gena i koristi se kao veza između izvornih baza koje za oznake gena koriste sinonime umesto izvornih simbola i pomoćnih baza prilikom dohvatanja nedostajućih atributa anotacijske datoteke.

Datoteka koja se koristi prilikom parsiranja baze `HUGO` može se pronaći pod nazivom koji započinje sa `hgnc_complete_set` i završava se datumom objavljivanja same datoteke, naziv datoteke koja se koristi u trenutku pisanja ovog rada je `hgnc_complete_set_2022-07-01.txt`. Može se besplatno preuzeti sa veb stranice putem sledećeg linka `http://ftp.ebi.ac.uk/pub/databases/genenames/hgnc/archive/monthly/tsv/` (poslednji put pristupano avgust 2022) gde su izlistane celokupne baze razvrstane po datumima objavljivanja na intervalu od mesec dana.

2.2.3 OBO

`OBO Foundry` predstavlja organizaciju koja za cilj ima da razvije porodicu interoperabilnih ontologija koje su logički dobro formirane i naučno tačne. Projekat `OBO` (eng. *The Open Biological and Biomedical Ontologies*) pokrenut je početkom 2000-ih, pošto je postalo jasno da postoji želja zajednice da se ontologije prošire izvan obima genetske ontologije kako bi se na širem planu pozabavile biološkim i biomedicinskim problemima [2]. `OBO` je dizajniran da organizuje i usmerava razvoj ontologija prema zajedničkim standardima i principima [20], omogućava modularnu kompoziciju ontologija obezbeđujući garancije tehničkog i naučnog kvaliteta. Postoji skup principa, koje treba da slede sve ontologije u okviru organizacije `OBO Foundry`. Na primer, baze `OBO` moraju biti besplatno dostupne, omogućavajući ponovnu upotrebu, a ontologije treba da budu u skladu sa zajedničkim standardima o tome kako su termini međusobno povezani. Trenutno, bazom `OBO` upravlja volonterski tim koji se sastoji od komiteta `OBO` i ovaj tim obavlja višestruke dužnosti, uključujući održavanje sajta, održavanje politike kompanije i stručno sakupljanje metapodataka o ontologiji [12]. Baza `OBO` kao jedna od pomoćnih baza ima značajnu ulogu prilikom pronalaženja vrednosti atributa anotacijske datoteke koji se odnose na bolest. Baza `OBO` ima mnoštvo atributa od kojih će pri kreiranju anotacijske datoteke biti korišćeni `id`, `name`, `def`, `synonym`, `xref` i `is_a` koji su prikazani u listingu 2.3, gde su zelenom bojom označene vrednosti koje se uzimaju za spomenute atribute. U narednoj listi biće dat opis atributa baze `OBO` kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. id - predstavlja identifikator bolesti i može se iskoristiti kao DOID atribut anotacijske datoteke.

2. name - predstavlja ime bolesti i može se iskoristiti kao Disease Name atribut anotacijske datoteke.

3. synonym - predstavlja sinonim bolesti i koristi se kao veza prema DOID atributu anotacijske datoteke.

4. xref - predstavlja Xref vrednost i koristi se kao veza radi preciznijeg pronalaženja nedostajućih atributa DOID i Disease Name anotacijske datoteke. Xref vrednosti koje se uzimaju u obzir prilikom mapiranja jesu GARD, ICD10CM, MEDDRA, MESH, OMIM i UMLS_CUI.

5. is_a - predstavlja generalizaciju same bolesti u vidu imena bolesti. Ovaj podatak je koristan prilikom parsiranja određenih izvornih baza koje sadrže vezu ka određenoj bolesti ali zahtevaju generalizaciju te bolesti.

6. def - predstavlja definiciju bolesti, koristi se prilikom inicijalizovanja pretraživača gde predstavlja jedan od kriterijuma po kome se vrši indeksiranje i pretraga.

Datoteka koja se koristi prilikom parsiranja baze OBO može se pronaći pod nazivom doid.obo i besplatno preuzeti sa veb stranice putem sledećeg linka https://obofoundry.org/ontology/doid.html (poslednji put pristupano avgust 2022).

2.2.4 RGD

Baza RGD (eng. *The Rat Genome Database*) osnovana je 1999. godine i brzo je postala glavno mesto za genetske, genomske, fenotipske i podatke vezane za bolesti dobijene iz istraživanja pacova. Pored toga, baza RGD se proširila tako da uključuje veliki korpus strukturiranih i standardizovanih podataka za osam vrsta (pacov, miš, čovek, činčila, bonobo, veverica sa 13 linija, pas i svinja). Veliki deo ovih podataka prikupljen je ručno od strane različitih naučnika, a ostali podaci su uvezeni u bazu RGD iz drugih baza [21].

Baza RGD je u velikoj meri po strukturi i sadržaju slična bazi OBO, pa je i sama primena baze RGD identična primeni baze OBO na nivou problema koji se rešava u ovom radu. Razlika između ove dve baze je u tome što baza RGD sadrži dodatni atribut alt_id čija je svrha identična kao svrha atributa xref koji je opisan u

```
[Term]
id: DOID:10606
name: blind loop syndrome
def: "An  intestinal disease characterized by a dysbalance of the
 bacterial flora of the small intestine, causing derangement to
 the normal physiological processes of digestion and absorption."
 [url:https //en.wikipedia.org/wiki/Blind_loop_syndrome,
  url:https //www.ncbi.nlm.nih.gov/pubmed/20572300]
subset: NCIthesaurus
synonym: "Bacterial overgrowth syndrome" EXACT []
synonym: "Blind loop syndrome" EXACT []
xref: ICD10CM:K90.2
xref: ICD9CM:579.2
xref: MESH:D001765
xref: NCI:C34431
xref: SNOMEDCT_US_2021_09_01:77225009
xref: UMLS_CUI:C0005750
is_a: DOID:5295 ! intestinal disease
```

Listing 2.3: Deo baze OBO

delu 2.2.3. Takođe, baza RGD sadrži veći broj bolesti od baze OBO, neke bolesti se poklapaju u obe baze ali postoje i bolesti koje su jedinstvene. Deo baze RGD dostupan je u listingu 2.4.

Datoteka koja se koristi prilikom parsiranja baze RGD može se pronaći pod nazivom RDO.obo i besplatno preuzeti sa veb stranice putem sledećeg linka https://download.rgd.mcw.edu/data_release/ontology_obo_files/disease/RDO.obo (poslednji put pristupano avgust 2022).

2.2.5 Ensembl

Baza Ensembl predstavlja bioinformatički projekat za organizovanje podataka u vezi sa genomskim sekvencama. Baza Ensembl je dostupna kao interaktivna veb stranica, skup datoteka i kao kompletan, prenosivi softverski sistem otvorenog koda za rukovanje genomima. Svi podaci su dati bez ograničenja, a kod je javno dostupan. Ensembl se nalazi na institutu EMBL-EBI (eng. *European Molecular Biology Laboratory's European Bioinformatics Institute*), koji se nalazi u sklopu kampusa Wellcome Genome u Hinxtonu, Velika Britanija [3].

```
[Term]
id: DOID:0050168
name: autoimmune polyendocrine syndrome type 2
alt_id: OMIM:269200
def: "An autoimmune polyendocrine syndrome that is characterized
 by abnormal functioning of the immune system that causes
 auto-reactivity against endocrine organs. It is more
 heterogeneous and has not been linked to one gene. (DO)"

 [http://en.wikipedia.org/wiki/
    Autoimmune_polyendocrine_syndrome_type_2 "DO"]
synonym: "APS2" EXACT []
synonym: "autoimmune polyendocrine type II" EXACT []
synonym: "autoimmune polyglandular type II" EXACT []
synonym: "multiple endocrine deficiency syndrome, type 2" EXACT []
synonym: "polyglandular autoimmune syndrome, type 2" EXACT []
synonym: "polyglandular deficiency syndrome, type 2" EXACT []
synonym: "Schmidt's syndrome" EXACT []
synonym: "Schmidt syndrome" EXACT []
xref: GARD:7611
is_a: DOID:14040 ! autoimmune polyendocrine syndrome
created_by: rgd
creation_date: 2017-10-27T00:00:00Z
```

Listing 2.4: Deo baze `RGD`

Prilikom kreiranja anotacijske datoteke koriste se dve datoteke baze `Ensembl`, koje su prikazane u vidu tabela 2.8 i 2.9. Baza `Ensembl` ima mnoštvo atributa od kojih će prilikom kreiranja anotacijske datoteke biti korišćeni `gene_stable_id`, `protein_stable_id` i `xref`. U narednoj listi biće dat opis atributa baze `Ensembl` za obe datoteke kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

1. `gene_stable_id` - predstavlja identifikator gena prema bazi podataka `Ensembl` i može se iskoristiti kao `Ensembl ID` atribut anotacijske datoteke.

2. `protein_stable_id` - predstavlja identifikator proteina i koristi se kao veza prema atributima `Ensembl ID`, `Entrez ID` i `UniProt ID` anotacijske datoteke.

3. `xref` - u zavisnosti od datoteke baze `Ensembl` može predstavljati `UniProt ID` ili `Entrez ID` atribut anotacijske datoteke.

gene_stable_id	transcript_stable_id	protein_stable_id	xref	...
ENSG00000160072	ENST00000673477	ENSP00000500094	83858	...
ENSG00000160072	ENST00000472194	-	83858	...
ENSG00000160072	ENST00000308647	ENSP00000311766	83858	...
ENSG00000142611	ENST00000511072	ENSP00000426975	63976	...
ENSG00000142611	ENST00000607632	-	63976	...
ENSG00000142611	ENST00000378391	ENSP00000367643	63976	...
ENSG00000157911	ENST00000288774	ENSP00000288774	5192	...
ENSG00000157911	ENST00000447513	ENSP00000407922	5192	...

Tabela 2.8: Deo baze `Ensembl-Entrez`

Datoteke koje se koriste prilikom parsiranja baze `Ensembl` mogu se besplatno preuzeti sa veb stranice putem sledećeg linka `https://ftp.ensembl.org/pub/current_tsv/homo_sapiens/` (poslednji put pristupano avgust 2022) i nazivaju se `Homo_sapiens.GRCh38.107.entrez.tsv` i `Homo_sapiens.GRCh38.107.uniprot.tsv`.

2.2.6 Orphanet Xref

Baza `Orphanet Xref` predstavlja suštinski jednu od datoteka baze `Orphanet` koja je detaljno opisana u delu 2.1.4, kao što se može videti u tom delu opisane su izvorne baze pa se lako može doći do zaključka da `Orphanet` predstavlja jedinu bazu koja se koristi kao izvorna i kao pomoćna. Baza `Orphanet Xref` koristi se kao pomoćna baza jer sadrži `Xref` veze koje su od značaja prilikom pronalaženja atributa `DOID` i `Disease Name` anotacijske datoteke.

Baza `Orphanet Xref` ima mnoštvo atributa od kojih će prilikom kreiranja anotacijske datoteke biti korišćeni `OrphaCode`, `Name` i `Xref` koji su prikazani u listingu 2.5. U narednoj listi biće dat opis atributa baze `Orphanet Xref` kao i opis njihove uloge prilikom kreiranja anotacijske datoteke:

gene_stable_id	transcript_stable_id	protein_stable_id	xref	...
ENSG00000160072	ENST00000673477	ENSP00000500094	Q5T9A4	...
ENSG00000160072	ENST00000673477	ENSP00000500094	Q5T9A4-1	...
ENSG00000160072	ENST00000308647	ENSP00000311766	A0A5K1VW56	...
ENSG00000142611	ENST00000378391	ENSP00000367643	Q9HAZ2	...
ENSG00000142611	ENST00000378391	ENSP00000367643	Q9HAZ2-2	...
ENSG00000142611	ENST00000514189	ENSP00000421400	D6RFY3	...
ENSG00000142611	ENST00000463591	ENSP00000476002	U3KQL6	...
ENSG00000157911	ENST00000288774	ENSP00000288774	A0A024R0A4	...

Tabela 2.9: Deo baze `Ensembl-UniProt`

1. `OrphaCode` - predstavlja jedinstveni numerički identifikator koji se koristi kao veza prema `Xref` vrednostima i jedinstveno određuje ime bolesti dato atributom `Name`.

2. `Name` - predstavlja ime bolesti koje je jedinstveno određeno atributom `OrphaCode` i može se iskoristiti kao `Disease Name` atribut anotacijske datoteke.

3. `Xref` - predstavlja vezu koja se koristi za pronalaženje nedostajućih atributa `DOID` i `Disease Name` anotacijske datoteke. Može imati vrednosti `GARD`, `ICD-10`, `MeSH`, `MedDRA`, `OMIM` i `UMLS`.

```
<Disorder id="5">
  <OrphaCode>93</OrphaCode>
   ...
  <Name lang="en">Aspartylglucosaminuria</Name>
   ...
  <ExternalReferenceList count="8">
    <ExternalReference id="104504">
      <Source>MedDRA</Source>
```

```
        <Reference>10068220</Reference>
       ...
      </ExternalReference>
      <ExternalReference id="104506">
        <Source>ICD-10</Source>
        <Reference>E77.1</Reference>
       ...
      </ExternalReference>
      <ExternalReference id="104502">
        <Source>UMLS</Source>
        <Reference>C0268225</Reference>
       ...
      </ExternalReference>
      <ExternalReference id="3655">
        <Source>OMIM</Source>
        <Reference>208400</Reference>
       ...
      </ExternalReference>
      <ExternalReference id="104500">
        <Source>MeSH</Source>
        <Reference>C538402</Reference>
       ...
      </ExternalReference>
      <ExternalReference id="127007">
        <Source>GARD</Source>
        <Reference>5854</Reference>
       ...
      </ExternalReference>
    </ExternalReferenceList>
   ...
  </Disorder>
```

Listing 2.5: Deo baze `Orphanet Xref`

Datoteka koja se koristi prilikom parsiranja baze `Orphanet Xref` može se pronaći pod nazivom `en_product1.xml` i besplatno preuzeti sa veb stranice putem sledećeg linka `http://www.orphadata.com/data/xml/en_product1.xml` (poslednji put

pristupano avgust 2022).

Glava 3

Pregled aplikacije GDA

Nakon pregleda izvora koji se koriste prilikom kreiranja anotacijske datoteke, u ovom poglavlju biće dat prikaz grafičkog korisničkog interfejsa aplikacije `GDA`, kao i opis funkcionalnosti koje pruža i kratak pregled korišćenja. Na kraju će biti prikazani i rezultati vezani za anotacijsku datoteku.

Neke od funkcionalnosti koje aplikacija podržava su: sortiranje, pretraga celokupne anotacijske datoteke, pretraga na nivou atributa, izvoz u određeni format (`PDF`, `Excel`, `CSV`) i štampanje. Na kraju procesa kreiranja anotacijske datoteke ona će biti sačuvana na disku pod nazivom `annotation_file.txt` u formatu opisanom u delu 1.3. Tako sačuvana datoteka predstavlja običan tekst nepogodan za svakodnevno korišćenje zbog svoje obimnosti. Potrebno je anotacijsku datoteku prikazati na način koji omogućava jednostavan pregled celokupne datoteke.

3.1 Grafički korisnički interfejs

Aplikacija `GDA` implementirana je u programskom jeziku `Python` i može se koristiti putem terminala. Pored toga, kao dodatak razvijen je i veb deo aplikacije koji služi da olakša korišćenje same aplikacije i pruži dodatnu udobnost. Prilikom pokretanja aplikacije početna stanica izgleda kao na slici 3.1.

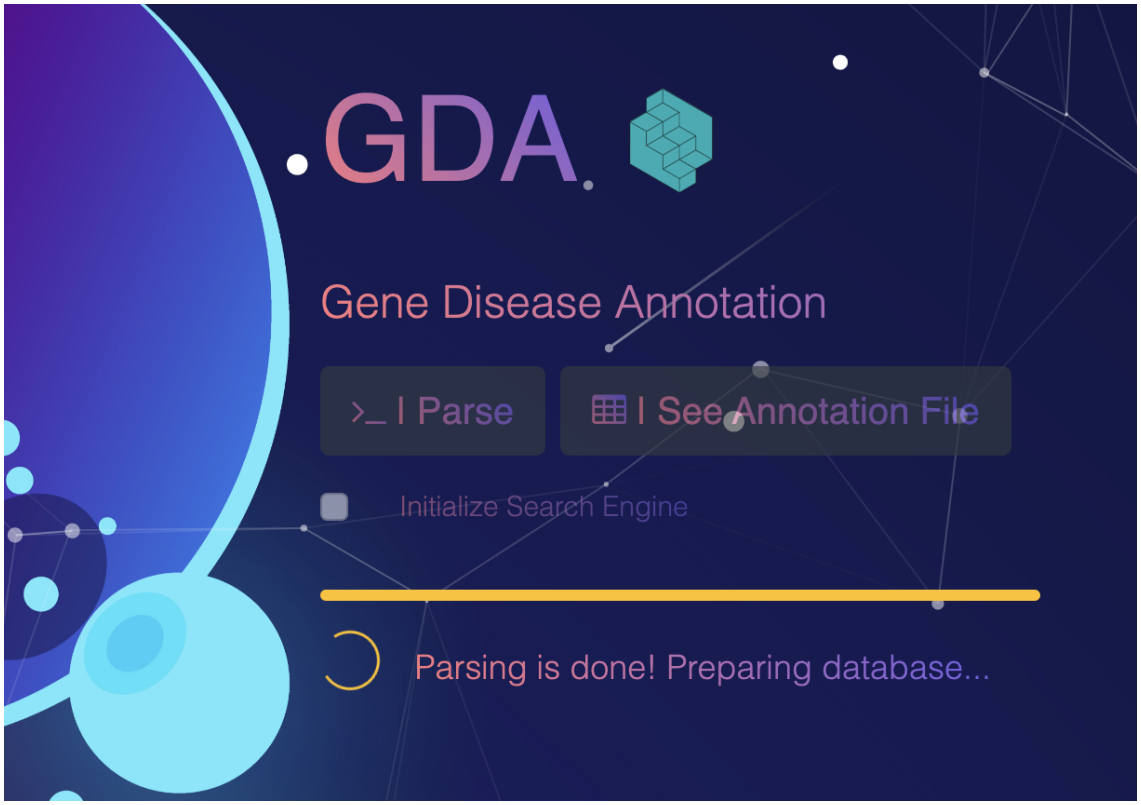
Početna stranica sadrži dva dugmeta `Parse` i `See Annotation File`, kao i *checkbox* `Initialize Search Engine`. Pritiskom na dugme `Parse` započinje se proces kreiranja i pripreme anotacijske datoteke za prikaz, gde će opis samog procesa i implementacija biti prikazani u narednom poglavlju. Tokom procesa kreiranja anotacijske datoteke korisnik može u svakom trenutku da vidi progres u vidu žute linije koja se popunjava. Nakon završenog procesa ispisuje se poruka da se priprema baza

Slika 3.1: Početna strana aplikacije

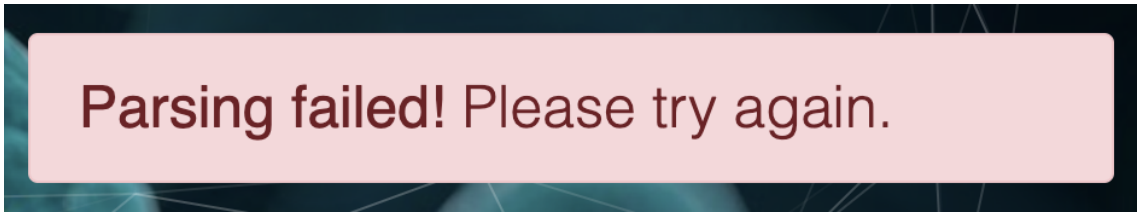
podataka i da je parsiranje uskoro gotovo. Na slici 3.2 prikazana je aplikacija tokom parsiranja. Nakon uspešno završenog parsiranja i pripremljene baze, korisnik se preusmerava na stranicu za prikaz anotacijske datoteke. Ukoliko je parsiranje neuspešno korisniku će biti prikazana poruka sa slike 3.3 i neće biti preusmeren na stanicu za prikaz anotacijske datoteke. Isto tako, pritiskom na dugme `See Annotation File` otvara se stranica koja sadrži prikaz anotacijske datoteke u vidu interaktivne tabele. Prikaz ove stranice dat je na slici 3.6. Eksplicitno navođenje da treba da se izvrši inicijalizacija pretraživača može se učiniti štikliranjem *checkbox*-a `Initialize Search Engine`. Inicijalizaciju pretraživača potrebno je učiniti ukoliko su dostupni novi podaci u bazama vezani za atribute `DOID` i `Disease Name`.

U podnožju stranice pored dugmadi koji se odnose na informacije o izvornom kodu i autoru aplikacije, nalazi se i dugme ℹ - `Help`, koje ima ulogu da dodatno objasni i prikaže funkcionalnosti aplikacije. Klikom na `Help` dugme otvara se prozor

unutar kog su napisana uputstva za korišćenje aplikacije. Svaka od stranica aplikacije ima zaseban `Help` prozor koji se odnosi na funkcionalnosti te stranice. `Help` prozori su prikazani na slikama 3.4 i 3.5.



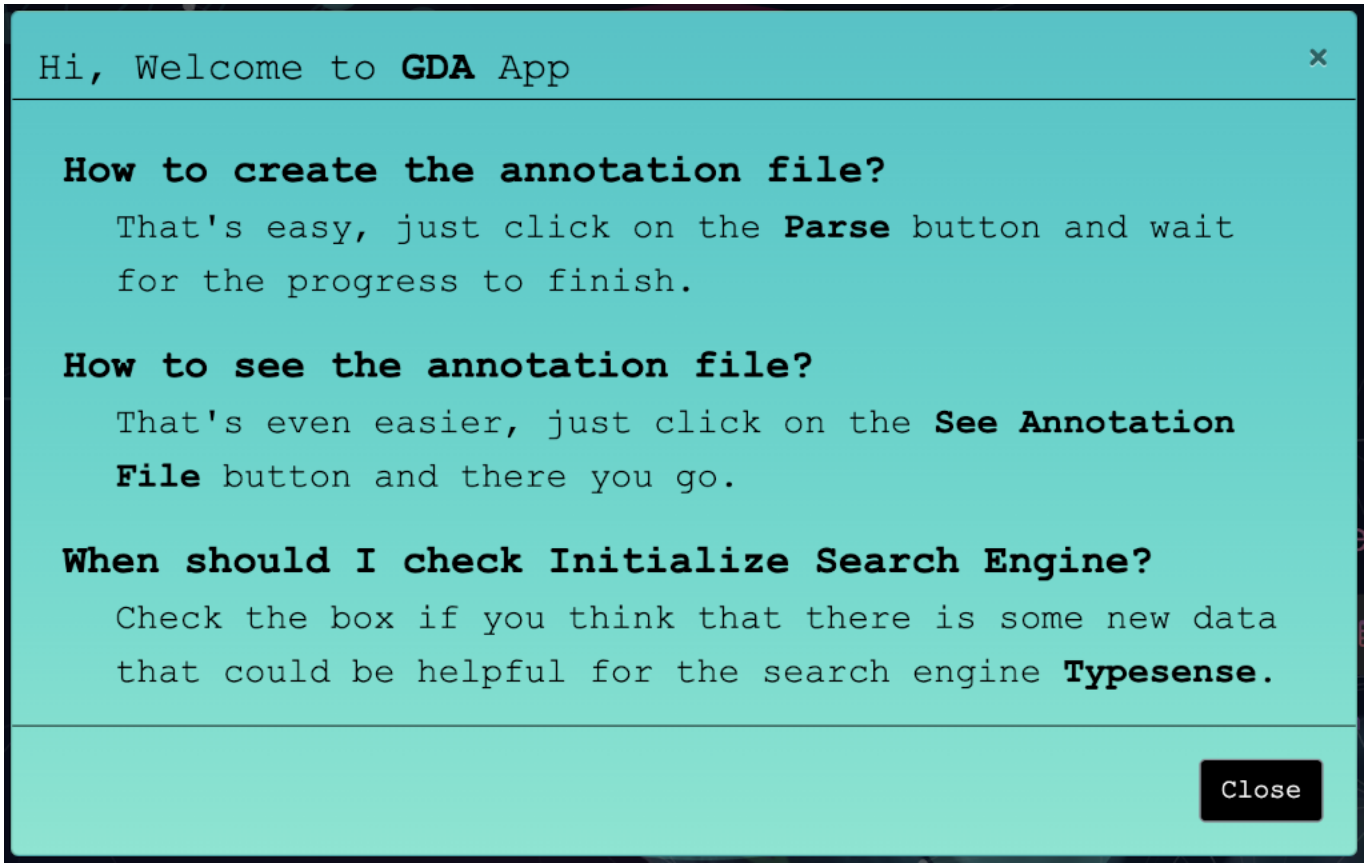
Slika 3.2: Prikaz aplikacije tokom kreiranja anotacijske datoteke



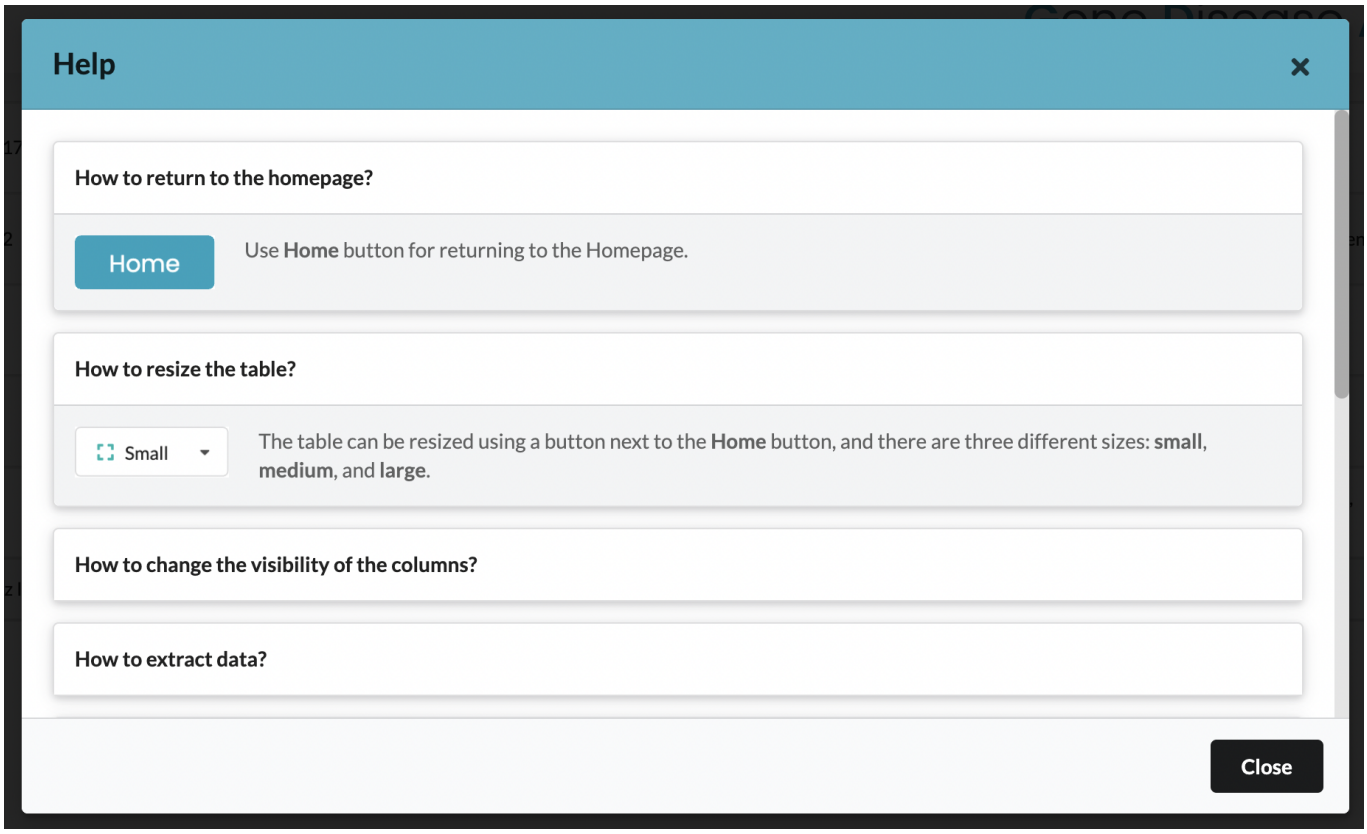
Slika 3.3: Poruka koja obaveštava korisnika o neuspešnom parsiranju

Stranica za prikaz anotacijske datoteke u svom zaglavlju sadrži dugme `Home` i padajući meni za odabir veličine same tabele. Pritiskom na dugme `Home` korisnik se preusmerava na početnu stranicu gde može ponovo započeti proces kreiranja i pripreme za prikaz anotacijske datoteke. Padajući meni za odabir veličine tabele sadrži tri elementa koji označavaju različite veličine `Small`, `Normal` i `Large`. U zavisnosti od veličine ekrana prikaz tabele se na ovaj način može prilagoditi tako da

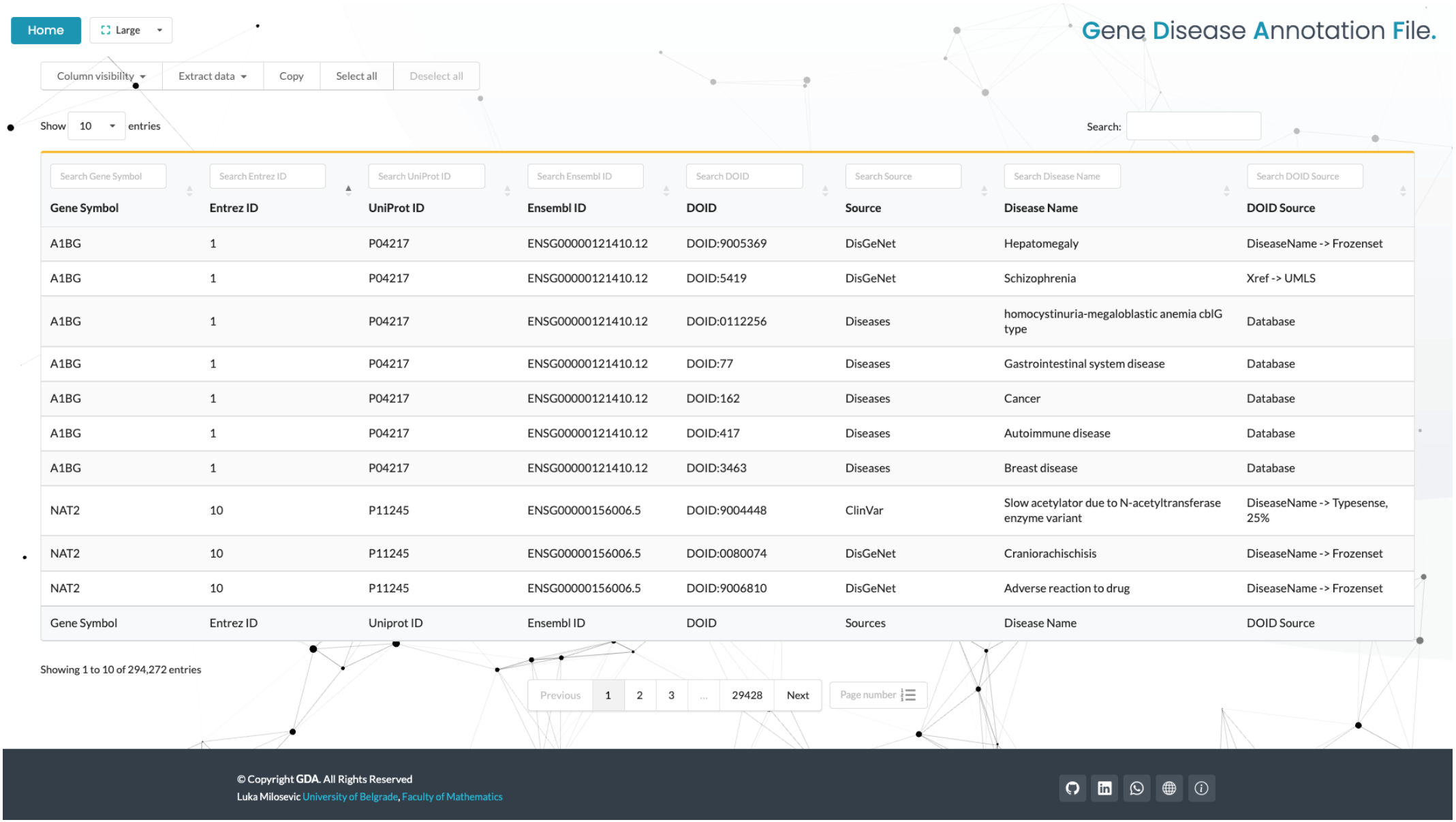
korisnik nema poteškoća prilikom korišćenja tabele. Naredni skup dugmadi predstavlja dodatne funkcionalnosti koje olakšavaju korišćenje interaktivne tabele za prikaz anotacijske datoteke.



Slika 3.4: `Help` prozor na početnoj stranici



Slika 3.5: `Help` prozor na stranici za prikaz anotacijske datoteke

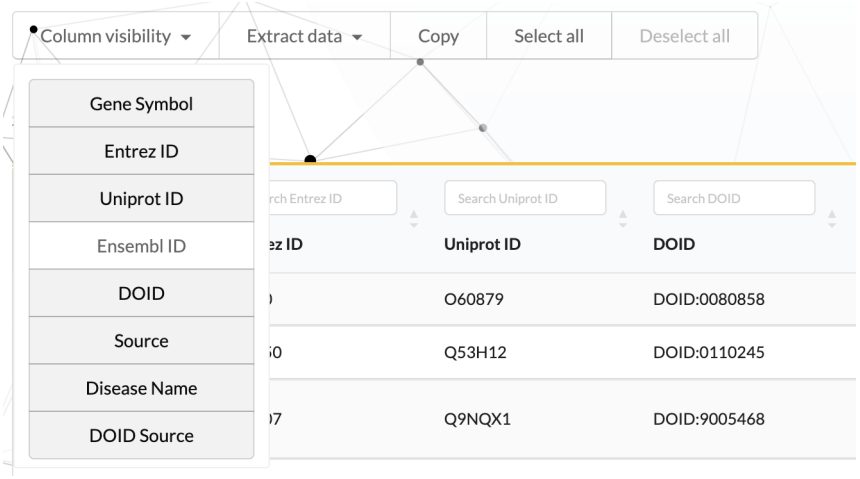


Slika 3.6: Stranica za prikaz anotacijske tabele

3.2 Funkcionalnosti interaktivne tabele

Prilikom korišćenja aplikacije korisniku je omogućeno da vrši različite manipulacije sa podacima na nivou interaktivne tabele. Dodatne funkcionalnosti koje aplikacija `GDA` pruža radi lakšeg manipulisanja podacima anotacijske datoteke su:

- `Column visibility` - predstavlja padajući meni, čiji su elementi predstavljeni kao nazivi kolona anotacijske datoteke. Odabirom nekog od naziva kolona može se upravljati vidljivošću same te kolone unutar interaktivne tabele. Na slici 3.7 dat je prikaz skrivene kolone `Ensembl ID`. Ova kolona trebalo bi da se nalazi između kolona `UniProt ID` i `DOID` koje su na prikazanoj slici jedna do druge.

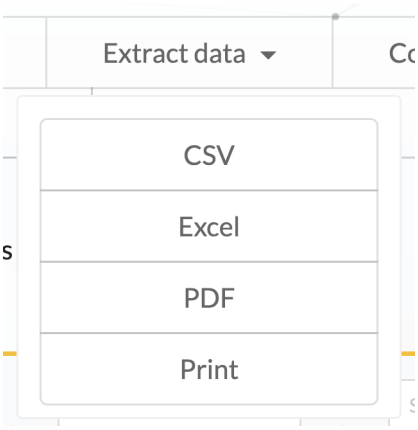


Slika 3.7: Funkcionalnost upravljanja vidljivošću kolona interaktivne tabele

- `Extract data` - predstavlja načine izvoza delova anotacijske datoteke u odabrani format. Interaktivna tabela pruža mogućnost selekcije više od jednog reda. Samim tim, delove anotacijske datoteke moguće je preuzeti pritiskom na dugme sa nazivom formata u kom će izabrani redovi biti sačuvani `CSV`, `Excel` ili `PDF`. Isto tako, deo anotacijske datoteke moguće je odštampati pritiskom na dugme `Print`. Ukoliko nije odabran nijedan red svaka od ove četiri naredbe biće primenjena na kompletnu stranu interaktivne tabele. Prikaz padajućeg menija gde se nalaze opisane funkcionalnosti nalazi se na slici 3.8.

- `Copy` - odabrani redovi biće kopirani, ukoliko nije odabran nijedan red biće kopirana kompletna strana interaktivne tabele.

- `Select all` - vrši se selekcija kompletne strane interaktivne tabele.



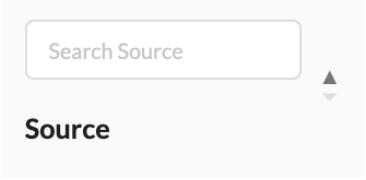
Slika 3.8: Funkcionalnost izdvajanja izabranih redova anotacijske datoteke

- `Deselect all` - vrši se deselekcija svih odabranih redova.

Naredne funkcionalnosti odnose se na interaktivnu tabelu i na to kako se vrši upravljanje podacima. Iznad interaktivne tabele nalazi se padajući meni koji se koristi za odabir broja redova koji će biti prikazani na jednoj strani unutar interaktivne tabele. Broj redova po strani može biti 10, 25, 50 i 100. Interaktivna tabela pruža mogućnost pretrage na nivou cele tabele i na nivou kolone. Mogućnost sortiranja kolona je takođe prisutna, pritiskom na zaglavlje kolone vrši se sortiranje redova po toj koloni uz indikator da li je sortiranje opadajuće ili rastuće. Prikaz opisanih funkcionalnosti dat je na slikama 3.9 , 3.10 i 3.11.

Slika 3.9: Funkcionalnost odabira broja redova

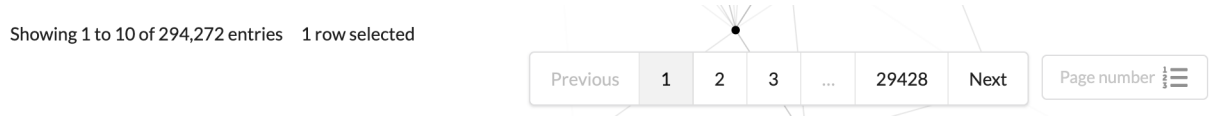
Slika 3.10: Pretraga na nivou tabele



Slika 3.11: Pretraga na nivou kolone i sortiranje

Preostale funkcionalnosti koje pruža interaktivna tabela su selekcija reda pritiskom na sam red ili selekcija nekoliko redova. Funkcionalnost selekcije redova bitna je jer pruža mogućnost izdvajanja samo izabranih redova u poseban dokument umesto celokupne stranice tabele. Prisutna je i funkcionalnost navigacije u vidu stranica zajedno sa dugmadima za odabir stranice, gde je moguće i uneti željeni broj stranice. Interaktivna tabela dinamički kreira stranice na osnovu broja redova po stranici i aktivne pretrage. Ispod interaktivne tabele nalazi se natpis koliko se redova prikazuje

od ukupnog broja redova i koliko je redova trenutno odabrano. Ove funkcionalnosti prikazane su na slici 3.12.



Slika 3.12: Prikaz broja redova i dugmadi za odabir stranice

3.3 Rezultati

Rezultati prikazani u ovom delu odnose se na vreme potrebno za kreiranje anotacijske datoteke, broj obrađenih redova i tačnost dobijenog atributa `DOID`.

3.3.1 Rezultati izvršavanja

Rezultati izvršavanja dobijeni su korišćenjem računara sa specifikacijama navedenim u listingu 3.1.

```
Model Name: MacBook Air
Chip: Apple M1
Total Number of Cores: 8 (4 performance and 4 efficiency)
Memory: 16 GB
Medium Type: SSD
Capacity: 512 GB
System Version: macOS 12.3.1 (21E258)
Kernel Version: Darwin 21.4.0
```

Listing 3.1: Specifikacije računara na kom je kreirana aplikacija `GDA`

Izvršavanje kreiranja anotacijske datoteke traje `191s` od toga:

- Čitanje i parsiranje izvornih i pomoćnih baza traje `15s`
- Pretprocesiranje uskladištenih podataka traje `81s`
- Pretraga nedostajućih atributa tj. finalno parsiranje traje `94s` od toga:
  - Finalno parsiranje baze `DisGeNet` traje `94s`.
  - Finalno parsiranje baze `COSMIC` traje `11s`.

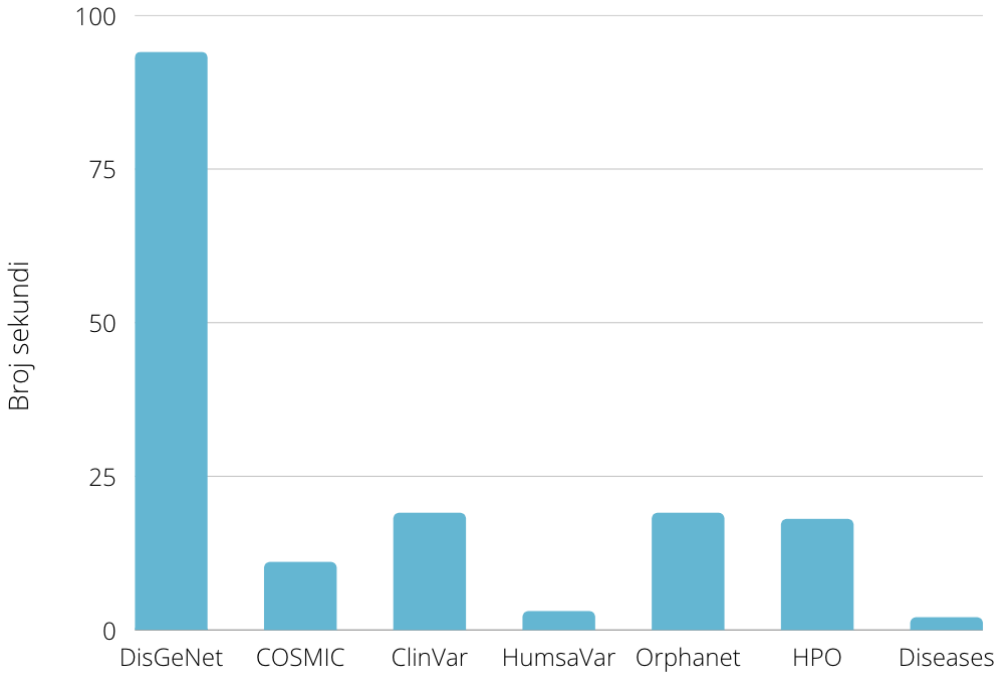
  - Finalno parsiranje baze `ClinVar` traje `19s`.
  - Finalno parsiranje baze `HumsaVar` traje `3s`.
  - Finalno parsiranje baze `Orphanet` traje `19s`.
  - Finalno parsiranje baze `HPO` traje `18s`.
  - Finalno parsiranje baze `Diseases` traje `2s`.

- Ostala izvršavanja traju `1s` (npr. skladištenje podataka anotacijske datoteke na disk).

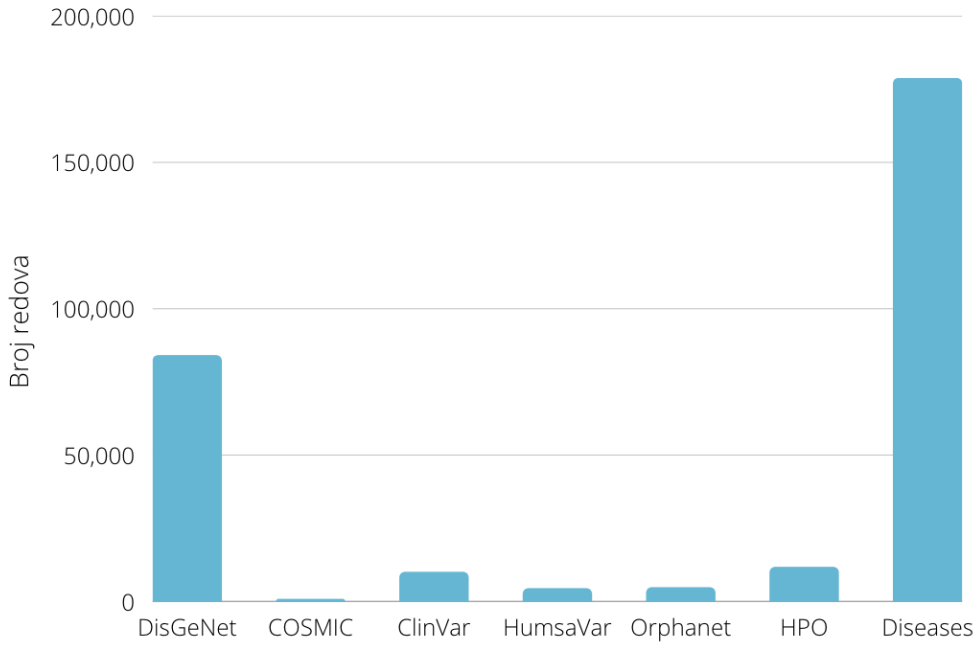
Ukupan broj entiteta koji se obrađuje tokom faze pretprocesiranja uskladištenih podataka iznosi `862263` entiteta spremnih za finalno parsiranje i za mapiranje tokom finalnog parsiranja. Finalno parsiranje započinje sa `294483` entiteta od toga:

- baza `DisGeNet` sadrži `84038` entiteta.
- baza `COSMIC` sadrži `783` entiteta.
- baza `ClinVar` sadrži `10024` entiteta.
- baza `HumsaVar` sadrži `4436` entiteta.
- baza `Orphanet` sadrži `4768` entiteta.
- baza `HPO` sadrži `11723` entiteta.
- baza `Diseases` sadrži `178711` entiteta.

Ovi rezultati prikazani su na slikama 3.13 i 3.14 kao stubičasti dijagrami.



Slika 3.13: Prikaz trajanja finalnog parsiranja po bazama



Slika 3.14: Prikaz broja entiteta sadržanih u bazama

Baza DisGeNet zahteva vremenski najduže finalno parsiranje što je očekivano jer predstavlja drugu najveću bazu po broju entiteta koje je potrebno obraditi. Prva baza po broju entiteta je baza Diseases ali ova baza sadrži vrednost atributa DOID čija pretraga predstavlja najkompleksniji korak. DisGeNet sadrži čak osam puta više entiteta od sledeće najveće baze.

Finalno parsiranje završava sa 294971 entiteta koji će biti smešteni unutar anotacijske datoteke. Ovaj broj razlikuje se u odnosu na početni jer su u slučaju poklapanja entiteta duplikati izbačeni, dok su neke vrednosti dodate iz razloga što neke baze mogu imati više vrednosti za jedan red (npr. baza HPO može na osnovu koda OMIM dobiti dodatne redove koji će biti sačuvani unutar anotacijske datoteke).

```
Reading sources elapsed time -> 14.875567197799683
  Preparing for parsing ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 862263/862263 0:01:20 0:00:00
Preparing parsing elapsed time -> 81.0408308506012
  Parsing   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 294483/294483 0:01:33 0:00:00
  DisGeNet  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 84038/84038   0:01:33 0:00:00
  Cosmic    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 783/783       0:00:11 0:00:00
  ClinVar   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 10024/10024   0:00:19 0:00:00
  HumsaVar  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 4436/4436     0:00:03 0:00:00
  Orphanet  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 4768/4768     0:00:19 0:00:00
  HPO       ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 11723/11723   0:00:18 0:00:00
  Diseases  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 100% 178711/178711 0:00:02 0:00:00
Parsing sources elapsed time -> 93.67263221740723
Total elapsed time -> 191.0115659236908
```

Slika 3.15: Izveštaj prikazan nakon pokretanja aplikacije GDA u terminalu (broj obrađenih entiteta po bazi i vreme potrebno za obradu)

Tokom izvršavanja aplikacije GDA putem terminala, korisnik može u svakom trenutku da vidi progres aplikacije - koliko entiteta iz koje baze je obrađeno, kao i koja baza se trenutno obrađuje. Prikaz ovakvog izvestaja na kraju izvrsavanja aplikacije je dat na slici 3.15.

3.3.2 Tačnost vrednosti atributa DOID

Zbog kompleksnosti pronalaženja vrednosti atributa DOID tokom izrade aplikacije kreirane su tri tabele koje prikazuju tačnost pronađenih vrednosti ovog atributa i opisane su u nastavku:

- Accuracy table - predstavlja tabelu koja kao redove sadrži imena izvornih baza, dok kao kolone sadrži potencijalne kategorije vrednosti atributa DOID. U ovoj tabeli sadržan je broj vrednosti atributa DOID razvrstan po kategorijama.

Ova tabela prikazuje koliko su pronađene vrednosti u nekoj bazi tačne. Prikaz rezultata dat je u tabeli 3.1.

Kategorije vrednosti mogu biti:

- `0%` - broj vrednosti koje su pronađene pomoću pretraživača sa tačnošću `0%`. Ukoliko je tačnost 0% to znači da ime bolesti koje pripada pronađenom `DOID`-u i ime bolesti koje se nalazi u redu koji se obrađuje nemaju nikakvo poklapanje. Ovo može da znači da je vrednost tog `DOID`-a pronađena na osnovu definicije bolesti.
- `(0, 25]` - broj vrednosti koje su pronađene pomoću pretraživača sa tačnošću u intervalu `(0, 25]`.
- `(25, 50]` - broj vrednosti koje su pronađene pomoću pretraživača sa tačnošću u intervalu `(25, 50]`.
- `(50, 75]` - broj vrednosti koje su pronađene pomoću pretraživača sa tačnošću u intervalu `(50, 75]`.
- `(75, 99]` - broj vrednosti koje su pronađene pomoću pretraživača sa tačnošću u intervalu `(75, 99]`.
- `100% (Xref)` - broj vrednosti koje su pronađene putem `Xref` veza, za tako pronađene vrednosti smatra se da je tačnost `100%`.
- `100% (Frozenset)` - broj vrednosti koje su pronađene putem rečnika korišćenjem `frozenset`-a za ključ, za tako pronađene vrednosti smatra se da je tačnost `100%`.
- `100% (Database)` - broj vrednosti koje su sadržane unutar baze koja se parsira, za tako pronađene vrednosti smatra se da je tačnost `100%`.
- `100% (Typesense)` - broj vrednosti koje su pronađene putem pretraživača `Typesense` ali sa tačnošću `100%`.
- `100% Total` - ukupan broj vrednosti sa tačnošću `100%`.
- `None` - broj nepronađenih vrednosti.
- `Total` - ukupan broj svih vrednosti.

- `Accuracy percentage table` - predstavlja identičnu tabelu kao prethodno opisanu s tim što umesto prebrojanih vrednosti sadrži rezultate u procentima. Prikaz rezultata dat je u tabeli 3.2.

- `Source table` - predstavlja tabelu koja kao redove sadrži imena izvornih baza, dok kao kolone sadrži vrednosti atributa `DOID Source`. U ovoj tabeli sadržan je broj vrednosti atributa `DOID` razvrstan po opisanim kolonama. Ova tabela prikazuje koliko je vrednosti atributa `DOID` u nekoj od izvornih baza dobijeno na osnovu nekog izvora. Prikaz rezultata dat je u tabeli 3.3.

  Izvori mogu biti:

  - `GARD` - broj vrednosti pronađenih na osnovu `Xref` vrednosti `GARD`.
  - `ICD10` - broj vrednosti pronađenih na osnovu `Xref` vrednosti `ICD10`.
  - `MeSH` - broj vrednosti pronađenih na osnovu `Xref` vrednosti `MeSH`.
  - `MedDRA` - broj vrednosti pronađenih na osnovu `Xref` vrednosti `MedDRA`.
  - `OMIM` - broj vrednosti pronađenih na osnovu `Xref` vrednosti `OMIM`.
  - `UMLS` - broj vrednosti pronađenih na osnovu `Xref` vrednosti `UMLS`.
  - `Frozenset` - broj vrednosti pronađenih na osnovu rečnika korišćenjem `frozenset`-a za ključ.
  - `Typesense` - broj vrednosti pronađenih na osnovu pretraživača `Typesense`.
  - `Database` - broj vrednosti sadržanih unutar baze koja se parsira.
  - `None` - broj nepronađenih vrednosti.
  - `Total` - ukupan broj svih vrednosti.

Među prikazanim rezultatima u tabelama 3.1 i 3.2, očekivano je da broj i procenat tačno pronađenih vrednosti atributa `DOID` bude najveći kada je u pitanju baza `Diseases`, zatim slede vrednosti pronađene korišćenjem `Xref` vrednosti i `frozenset`-a i na kraju korišćenjem pretraživača. Razlog kod baze `Diseases` je u tome što se za nju ne vrši pretraga atributa `DOID` jer je on sadržan unutar te baze, a pritom je i najobimnija baza. Interesantnije je koji je razlog kod sledeća dva načina: prvo se pokušava pretraga pomoću `Xref` vrednosti, a zatim pomoću `frozenset`-a čime je pretrazi pomoću `Xref` vrednosti data blaga prednost. Neke od baza (npr. `COSMIC i HPO`) nemaju veliki broj tačno pronađenih vrednosti `DOID`-a zbog specifičnosti samih naziva bolesti (npr. bolesti *AML, JMML* i *MDS* u bazi `COSMIC`) i toga što te baze nemaju kao atribut neku od `Xref` vrednosti. Ukupan procenat tačno pronađenih vrednosti atributa `DOID` od 97% predstavlja veoma dobar rezultat koji je posebno poboljšan dodavanjem `Xref` vrednosti kao jedan od načina pretrage ovog atributa. Ovo poboljšanje je vidljivo na osnovu procenta tačno pronađenih atributa `DOID` na

osnovu ove tehnike koji iznosi 20% i nalazi se odmah iza načina pretrage ovog atributa kada je atribut `DOID` sadržan unutar baze koja se parsira čiji procenat iznosi 61%.

Među rezultatima prikazanim u tabeli 3.3, izuzev rezultata vezanih za `Diseases` bazu, očekivano je da najveći broj pronađenih vrednosti atributa `DOID` budu korišćenjem `Xref` vrednosti i to specifično `UMLS` i `OMIM`. Ove dve veze dominiraju iz razloga što ih mnoge baze sadrže kao atribute. Interesantno je da nijedna vrednost atributa `DOID` nije pronađena pomoću `MedDRA` veze. Takvo ponašanje je realno jer su vezama dodeljeni prioriteti na osnovu pregleda koje od veza se nalaze unutar baza koje se parsiraju, gde je `MedDRA` veza manjeg prioriteta u odnosu na ostale jer se ne nalazi unutar izvornih baza.

3.3.3 Specifični redovi anotacijske datoteke

Anotacijska datoteka zamišljena je da kao redove sadrži veze gen-bolest i dodatne informacije o konkretnim genima i bolestima. Međutim, postoje redovi u anotacijskoj datoteci koji nisu kompletni. U sledećoj listi biće opisano kakve to anomalije anotacijska datoteka može da sadrži i koji su razlozi tome:

- **Nedostajuće vrednosti nekih atributa** - ovaj oblik podrazumeva da je prisutan barem jedan atribut vezan za gen i barem jedan atribut vezan za bolest, a nedostaju neki od preostalih atributa. Razlog nedostatka vrednosti ovih atributa je što ostale baze ne sadrže tu informaciju ili se takva informacija drugačije naziva u drugim bazama.

- **Nedostajuće vrednosti svih atributa o genu** - ovaj oblik podrazumeva da su atributi vezani za gen nepoznati. Ovaj problem je povezan sa bazom `Diseases` koja sadrži određene nekonzistentnosti opisane u delu 4.3.1. Dakle, ukoliko se umesto atributa `Gene Symbol` nalazi identifikator proteina i pritom ako preuzimanje identifikatora gena pomoću identifikatora proteina nije moguće, informacija vezana za gen neće biti pronađena i ostaće samo atributi vezani za bolest. Da informacija o genu ne bi izostala u potpunosti, na mestu atributa `Gene Symbol` biće sačuvan identifikator proteina. Anotacijska datoteka u svom izvornom obliku, kao tekstualna datoteka, sadržaće dodatni tag `<PROTEIN_ID>` koji će se nalaziti uz vrednost identifikatora proteina (npr. `<PROTEIN_ID>ENSP00000387888`) koja je sačuvana na mestu atributa `Gene`

Symbol. Unutar interaktivne tabele ovako sačuvani identifikatori proteina sadržaće znak # uz svoju vrednost (npr. #ENSP00000387888). Takođe, prelaskom miša preko ovih vrednosti biće ispisano Ensembl Protein ID. Anotacijska datoteka trenutno sadrži 109 ovakvih redova.

- **Nedostajuće vrednosti svih atributa o bolesti** - ovaj oblik podrazumeva da su atributi vezani za bolest nepoznati. Ovaj problem je povezan sa bazom HPO koja sadrži kao atribute kodove OMIM i ORPHA. Pomoću ovih kodova vrši se pretraga imena bolesti i atributa DOID za koju nije garantovano da će uvek biti uspešna sa konkretnim rezultatima. Ukoliko pretraga nije uspešna redovi baze HPO neće imati informaciju o bolesti. Da informacija o bolesti ne bi izostala u potpunosti, na mestu atributa Disease Name biće sačuvan jedan od kodova OMIM ili ORPHA. Anotacijska datoteka u svom izvornom obliku, kao tekstualna datoteka, sadržaće dodatni tag <OMIM> ili <ORPHA> koji će se nalaziti uz vrednost kodova OMIM ili ORPHA (npr. <OMIM>614129 ili <ORPHA>98905) i biće sačuvani na mestu atributa Disease Name. Unutar interaktivne tabele ovako sačuvani OMIM ili ORPHA kodovi sadržaće dodatne oznake OMIM: i ORPHA: uz svoju vrednost (npr. OMIM:614129 ili ORPHA:98905). Takođe, prelaskom miša preko ovih vrednosti biće ispisano OMIM Code ili ORPHA Code, u zavisnosti od vrste koda. Anotacijska datoteka trenutno sadrži 2056 ovakvih redova.

	0%	(0, 25]	(25, 50]	(50, 75]	(75, 99]	100% Xref	100% Frozenset	100% Database	100% Typesense	Total 100%	None	Total
ClinVar	9	208	261	135	0	8180	1229	0	0	9409	2	10024
Cosmic	19	97	221	47	0	0	326	0	0	326	73	783
DisGeNet	368	1572	1843	1755	0	31416	46877	0	0	78293	207	84038
Diseases	0	0	0	0	0	0	0	178710	0	178710	0	178710
HPO	0	9	17	4	0	9816	309	0	0	10125	2057	12212
HumsaVar	0	3	4	6	0	4390	33	0	0	4423	0	4436
Orphanet	17	93	132	69	0	4002	446	0	0	4448	9	4768
Total	413	1982	2478	2016	0	57804	49220	178710	0	285734	2348	294971
Total %	0.14	1	1	1	0	20	17	61	0	97	1	100

Tabela 3.1: Prikaz tabele `Accuracy table`

	0%	(0, 25]	(25, 50]	(50, 75]	(75, 99]	100% Xref	100% Frozenset	100% Database	100% Typesense	Total 100%	None	Total
ClinVar	0.09	2	3	1	0	82	12	0	0	94	0.02	10024
Cosmic	2	12	28	6	0	0	42	0	0	42	9	783
DisGeNet	0.438	2	2	2	0	37	56	0	0	93	0.246	84038
Diseases	0	0	0	0	0	0	0	100	0	100	0	178710
HPO	0	0.074	0.139	0.033	0	80	3	0	0	83	17	12212
HumsaVar	0	0.068	0.09	0.135	0	99	1	0	0	100	0	4436
Orphanet	0.357	2	3	1	0	84	9	0	0	93	0.189	4768
Total %	0.14	1	1	1	0	20	17	61	0	97	1	100

Tabela 3.2: Prikaz tabele `Accuracy percentage table`

	GARD	ICD10	MeSH	MedDRA	OMIM	UMLS	Frozenset	Typesense	Database	None	Total
ClinVar	0	0	0	0	**5831**	2349	1229	613	0	2	10024
Cosmic	0	0	0	0	0	0	326	**384**	0	73	783
DisGeNet	0	0	0	0	0	31416	**46879**	5536	0	207	84038
Diseases	0	0	0	0	0	0	0	0	**178710**	0	178710
HPO	365	94	382	0	**7892**	1083	309	30	0	2057	12212
HumsaVar	0	0	0	0	**4390**	0	33	13	0	0	4436
Orphanet	488	179	494	0	**1528**	1313	446	312	0	8	4768
Total	853	273	876	0	19641	36161	49220	6888	178710	2348	294971
Total%	0.289	0.093	0.297	0	7	12	17	2	61	1	100

Tabela 3.3: Prikaz tabele `Source table`

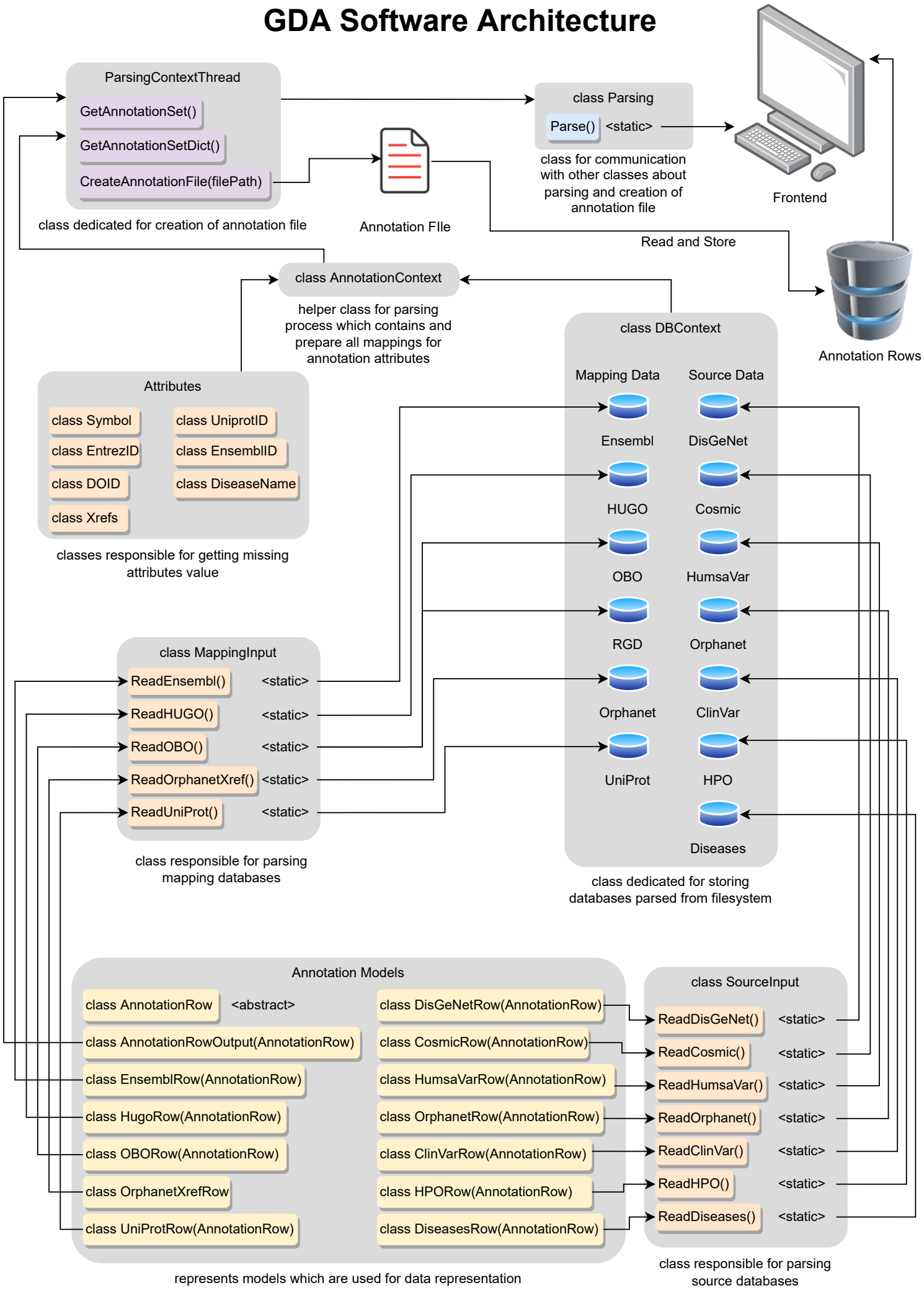
Glava 4

Opis i arhitektura aplikacije GDA

Nakon pregleda aplikacije `GDA`, u ovom poglavlju biće opisana implementacija aplikacije, njena arhitektura kao i korišćene tehnologije. Cilj ove aplikacije je kreiranje anotacijske datoteke na osnovu više različitih, neuniformnih izvora i njen prikaz u funkcionalnom obliku. Aplikacija `GDA` kreirana je u formi veb aplikacije otvorenog koda što doprinosi njenoj portabilnosti. Portabilnost se postiže postupkom dokerizovanja[1] same aplikacije, koja se uz pomoć `Docker Engina`-a pokreće na operativnim sistemima koje `Docker` podržava (npr. `Windows`, `MacOS` i `Linux`).

Aplikacija se sastoji iz dva dela, interfejsa aplikacije (eng. *frontend*) i serverskog dela aplikacije (eng. *backend*). Serverski deo aplikacije odnosi se na celokupan proces dobijanja anotacijske datoteke od početnog čitanja baza do finalnog parsiranja i mapiranja, dok interfejs aplikacije predstavlja veb interfejs koji se sastoji od početne stranice i stranice koja sadrži anotacijsku datoteku prikazanu u formi interaktivne tabele. Za interfejs aplikacije korišćen je `Django Framework`, dok je serverski deo aplikacije implementiran u programskom jeziku `Python`. Na slici 4.1 dat je grub prikaz arhitekture aplikacije gde su prikazani svi najznačajniji delovi koji će u narednim poglavljima biti prikazani i objašnjeni detaljnije. U ovom poglavlju akcenat će biti na serverskom delu aplikacije dok će interfejs aplikacije biti opisan na višem nivou apstrakcije u odnosu na serverski deo aplikacije.

[1]Dokerizovanje (eng. *dockerizing*) predstavlja pojam kreiranja `Docker` softverskog kontejnera (eng. *Docker container*), koji sadrži sve procedure neophodne za izvršavanje dokerizovane aplikacije.



Slika 4.1: Arhitektura aplikacije GDA

4.1 Kreiranje anotacijske datoteke

U uvodnom delu poglavlja prikazana je arhitektura aplikacije na najvišem nivou apstrakcije gde su predstavljeni elementi koji učestvuju u procesu kreiranja anotacijske datoteke. U ovom delu pored arhitekture biće opisan i proces kreiranja anotacijske datoteke što predstavlja srž aplikacije `GDA` za prikaz veze gena i bolesti. U nastavku poglavlja delovi arhitekture biće objašnjeni detaljnije kroz faze razvoja aplikacije `GDA`. Proces kreiranja anotacijske datoteke sastoji se iz tri faze:

- Čitanje i parsiranje izvornih i pomoćnih baza
- Pretprocesiranje uskladištenih podataka
- Pretraga nedostajućih atributa i kreiranje anotacijske datoteke

Svaka od ove tri faze biće detaljno objašnjena u nastavku sa primerima izvornog koda procesa kreiranja anotacijske datoteke.

4.2 Puno pretraživanje teksta i pretraživač Typesense

Pre nego što bude opisan proces kreiranja anotacijske datoteke bitno je predstaviti program koji je integrisan u projekat aplikacije `GDA` kao program otvorenog koda. Program koji je integrisan u sam projekat naziva se `Typesense` i koristi se za pretraživanje zadatih pojmova.

Problem kreiranja anotacijske datoteke prvenstveno predstavlja problem zasnovan na manipulaciji tekstom. Prilikom izrade anotacijske datoteke neophodno je vršiti pretraživanje tekstualnih baza podataka. Ovakva vrsta pretrage može biti zahtevna zbog obima baza koje se koriste. Većina atributa sadrži identifikatore koje je jednostavno pretražiti i uporediti jer su jedinstveni. Pretraga atributa koji sadrže informacije u tekstualnom formatu koji nije uniforman predstavlja izazov koji je potrebno rešiti. Deo veze koja se odnosi na bolest sastoji se iz dva atributa, `DOID` i `Disease Name`. Jedan od načina pronalaženja atributa `DOID` jeste korišćenjem atributa `Disease Name`. Atribut `Disease Name` kao što je već pomenuto sadrži ime bolesti koje nije jedinstveno i predstavlja nisku koju nije jednostavno pronaći u nekom velikom skupu.

Tehnika punog pretraživanja teksta nameće se kao pogodno rešenje pretraživanja obimnih tekstova. Ova tehnika odnosi se na pretraživanje jednog ili nekoliko dokumenata ili tekstualne kolekcije unutar baze podataka. Prilikom punog pretraživanja teksta, pretraživač ispituje sve reči u svakom sačuvanom dokumentu dok pokušava da pronađe podudaranje sa kriterijumima pretrage. Mnoge aplikacije i veb sajtovi svoje pretrage zasnivaju na ovoj tehnici. Puno pretraživanje teksta namenjeno je pretraživanju velikih količina teksta. Na primer, veb pretraživač će koristiti ovu tehniku da traži ključne reči na svim veb stranicama koje je indeksirao. Ključ ove tehnike je indeksiranje. Kada se radi sa malim brojem dokumenata, moguće je da pretraživač direktno skenira dokumenta sa svakim upitom i takva strategija se zove serijsko skeniranje (eng. *serial scanning*). Međutim, kada je broj dokumenata za pretragu potencijalno veliki ili je količina upita za pretragu velika, problem pretrage celog teksta se često deli na dva zadatka: indeksiranje i pretraživanje. Faza indeksiranja će skenirati tekst svih dokumenata i napraviti listu termina za pretragu koji se nazivaju indeksi. U fazi pretrage, kada se obavlja određeni upit, ono na šta se vrši referenca je samo indeks, a ne tekst originalnih dokumenata.

Pretraživač `Typesense` koristi tehniku punog pretraživanja teksta i stoga predstavlja pogodno rešenje za intenzivnu pretragu kakva je ovde neophodna. Ovaj pretraživač je otvorenog koda, otporan je na greške u kucanju i optimizovan je za trenutne pretrage koje ne traju duže od 50ms [24]. `Typesense` pruža podršku za različite programske jezike kroz klijentske biblioteke (eng. *client libraries*). Prilikom kreiranja anotacijske datoteke korišćena je klijentska biblioteka za programski jezik `Python` i zvanična doker slika (eng. *docker image*) celokupnog pretraživača.

4.3 Čitanje i parsiranje izvornih i pomoćnih baza

Proces kreiranja anotacijske datoteke započinje fazom čitanja i parsiranje izvornih i pomoćnih baza. Datoteke baza sačuvane su u formatima `.csv`, `.tsv`, `.dat`, `.txt`, `xml` i `.obo`, svaki od ovih formata zahteva različit pristup parsiranju, gde se za određene formate koriste biblioteke programskog jezika `Python`, dok preostale zahtevaju specifičan način parsiranja prilagođen formatu same baze. Postoje dve klase zadužene za čitanje i parsiranje, `class SourceInput` zadužena za izvorne baze i `class MappingInput` zadužena za pomoćne baze, koje sadrže statički metod za svaku bazu pomoću kog se vrši čitanje i parsiranje iste te baze. Prilikom parsiranja pročitani podaci čuvaju se kao anotacijski modeli, koji predstavljaju klase

dizajnirane po ugledu na podatke koje određena baza sadrži. Klasa `AnnotationRow` predstavlja anotacijski model koji sadrži sve atribute anotacijske datoteke i koristi se kao osnova za ostale anotacijske modele.

```
class AnnotationRow:
    def __init__(self, symbol, entrezID, uniprotID, ensemblID,
                 doid, source, diseaseName):
        self.symbol = symbol
        self.entrezID = entrezID
        self.uniprotID = uniprotID
        self.ensemblID = ensemblID
        self.doid = doid
        self.source = source
        self.diseaseName = diseaseName

    def __eq__(self, other):
        return (isinstance(other, self.__class__) and
                other.symbol == self.symbol and
                other.entrezID == self.entrezID and
                other.uniprotID == self.uniprotID and
                other.ensemblID == self.ensemblID and
                other.doid == self.doid and
                other.source == self.source and
                other.diseaseName == self.diseaseName)

    def __hash__(self):
        return hash((self.symbol, self.entrezID,
            self.uniprotID, self.ensemblID,
            self.doid, self.source, self.diseaseName))
```

Listing 4.1: Osnovni anotacijski model `AnnotationRow`

Izvorne i pomoćne baze sadrže atribute koji su od koristi za kreiranje anotacijske datoteke, isto tako sadrže i atribute koji nemaju nikakav značaj za anotacijsku datoteku. Takve atribute treba izdvojiti i zadržati one koji će se naći u anotacijskoj datoteci kao i one koji će pomoći prilikom kreiranja anotacijske datoteke. Pojedinačne baze sadrže neke od atributa anotacijske datoteke, tako da anotacijski modeli tih baza nasleđuju osnovni anotacijski model `AnnotationRow` što je prikazano u kodu 4.2 na primeru anotacijskog modela baze `Orphanet`.

Kada se odbace atributi koji nemaju značaj za samu anotacijsku datoteku, nailazi se na problem redundantnih podataka koji se javlja iz razloga što je broj kolona manji, a broj redova je ostao nepromenjen. Problem redundantnosti podataka otklanja se korišćenjem strukture podataka `OrderedSet` koja se ponaša kao skup tj. ne sadrži duplikate. Metode `__eq__` i `__hash__` klase `AnnotationRow` prikazane u kodu 4.1 koriste se prilikom upoređivanja elemenata anotacijskog modela `AnnotationRow`, kao i elemenata čiji anotacijski model nasleđuje osnovni anotacijski model. Ove metode će biti pozvane prilikom dodavanja novih elemenata u strukturu `OrderedSet`. Elementi koji se već nalaze unutar te strukture neće biti ponovo dodati i time se rešava problem redundantnih podataka.

```
class OrphanetRow(AnnotationRow):
    def __init__(self, symbol, uniprotID, ensemblID, diseaseName,
            orpha):
        super(OrphanetRow, self).__init__(symbol, None, uniprotID,
            ensemblID, None, "Orphanet", diseaseName)
        self.orpha = orpha

    def __eq__(self, other):
        return super(OrphanetRow, self).__eq__(other)
            and self.orpha == other.orpha

    def __hash__(self):
        return hash((self.symbol, self.uniprotID, self.ensemblID,
            self.diseaseName, self.orpha))

    def __str__(self):
        return super(OrphanetRow, self).__str__() + '\t'
            + str(self.orpha)
```

Listing 4.2: Anotacijski model baze `Orphanet`

Nakon završenog procesa čitanja i parsiranja, podaci se dalje prosleđuju u vidu strukture `OrderedSet` koja u zavisnosti od baze koja se parsira sadrži elemente čiji je tip anotacijski model baze koja se parsira. Samim tim, ta struktura predstavlja povratnu vrednost statičkih metoda koji se koriste za čitanje i parsiranje. Dat je primer statičke metode zadužene za čitanje i parsiranje baze `DisGeNet` u kodu 4.3.

U nastavku će biti opisano parsiranje za one baze koje sadrže specifične tehnike

parsiranja ili postoje određeni uslovi pod kojima je vršeno parsiranje, dok će za preostale baze biti spomenute biblioteke koje se koriste prilikom čitanja i specifičnosti parsiranja formata kom pripadaju. U tabeli 4.1 dat je sumiran prikaz baza sa tehnikama koje koriste.

Baza	**Način parsiranja**
DisGeNet	Pandas
COSMIC	Pandas
ClinVar	Pandas
Diseases	Pandas
Ensembl	Pandas
HUGO	Pandas
Orphanet	ElementTree
Orphanet Xref	ElementTree
OBO	pronto
RGD	pronto
HumsaVar	Nekonvencijalno parsiranje
HPO	Nekonvencijalno parsiranje
UniProt	Nekonvencijalno parsiranje

Tabela 4.1: Tehnike čitanja i parsiranja baza

4.3.1 Parsiranje bibliotekom Pandas

Biblioteka Pandas predstavlja brzu i fleksibilnu biblioteku otvorenog koda za programski jezik Python koja pruža visoke performanse za ceo proces analize poda-

taka na jednostavan i intuitivan način. Baze koje se čuvaju u formatima `.csv` i `.tsv`[2] pogodne su za parsiranje korišćenjem biblioteke `Pandas`. Biblioteka `Pandas` sadrži metod `read_csv` koji se koristi prilikom čitanja baza `DisGeNet`, `COSMIC`, `ClinVar`, `Diseases`, `Ensembl` i `HUGO`. Metod `read_csv` poziva se sa argumentima koji označavaju putanju do datoteke na disku, separatorom koji se koristi prilikom čitanja same datoteke i tipom podatka koji će biti pročitan. Korišćenje biblioteke `Pandas` prilikom čitanja i parsiranja može se videti na primeru baze `DisGeNet` u kodu 4.3.

```
@staticmethod
def ReadDisGeNet(filePath=DISGENET_PATH):
    disGeNetData = pd.read_csv(filePath, sep='\t', dtype=str)
    disGeNetData = disGeNetData[["geneId", "geneSymbol",
        "diseaseId", "diseaseName"]]
    disGeNetData = disGeNetData.to_numpy()

    disGeNetSet = OrderedSet()
    for row in disGeNetData:
        symbol = CheckNan(row[1])
        entrezID = CheckNan(row[0])
        umls = CheckNan(row[2])
        diseaseName = CheckNan(row[3])
        disGeNetSet.add(DisGeNetRow(symbol, entrezID,
            diseaseName, umls))

    return disGeNetSet
```

Listing 4.3: Statički metod za čitanje i parsiranje baze `DisGeNet`

Prilikom parsiranja baze `Diseases` bitno je napomenuti da se čuvaju samo oni redovi koji nemaju zastareli identifikator bolesti tj. atribut `DOID` nije *obsolete*. Dodatno, u okviru baze `Diseases` (deo je prikazan u tabeli 2.7) može se uočiti problem nekonzistentnosti kolona `Gene Identifier` i `Gene Name`. Nekonzistentnost se može primetiti u prvom i poslednjem redu gde su vrednosti obe kolone identične. Problem koji se javlja je taj da pojedini redovi u ove dve kolone mogu sadržati identične vrednosti i to tako da neki redovi sadrže u obe kolone identifikator proteina ili simbol

[2]Razlika između `.csv` i `.tsv` formata je u tome što su polja kod `.tsv` formata razdvojena tabovima, a kod `.csv` formata se koristi zapeta. Obe vrste formata koriste novi red kao separator za pojmove.

gena. Ovaj problem se otklanja tako što ukoliko se u obe kolone nađe simbol gena, identifikator proteina sačuvaće se kao vrednost `None` i obrnuto.

4.3.2 Parsiranje bibliotekom ElementTree

`ElementTree` predstavlja jednostavnu i efikasnu biblioteku programskog jezika `Python` koja se koristi za čitanje, pravljenje ili parsiranje `XML`[3] datoteka. `ElementTree` u svom radu razbija `XML` dokument u strukturu stabla kojom je lako manipulisati. Parsiranje se započinje metodom `parse` koja kao argument prima putanju do datoteke na disku i kao povratnu vrednost daje strukturu stabla. Parsiranje se nastavlja različitim metodama koje pruža biblioteka `ElementTree` za lako manipulisanje strukturom stabla. Korišćenje biblioteke `ElementTree` može se videti na delu čitanja i parsiranja baze `Orphanet` u kodu 4.4.

```
@staticmethod
def ReadOrphanet(filePath=ORPHANET_PATH):
    orphanetSet = OrderedSet()
    tree = et.parse(filePath)
    root = tree.getroot()
    disorderList = root.find("DisorderList")
    for disorder in disorderList:
        disorderType = disorder.find("DisorderType").find("Name")
            .text.strip()
        if disorderType != "Disease":
            continue
        ...

    return orphanetSet
```

Listing 4.4: Deo statičkog metoda za čitanje i parsiranje baze `Orphanet`

Baze koje koriste ovu biblioteku tokom parsiranja su `Orphanet` i `Orphanet Xref` koje su sačuvane u `.xml` formatu. Za sam proces parsiranja ovih baza bitno je napomenuti da se uzimaju oni entiteti koji za `DisorderType` atribut imaju vrednost

[3] `XML` (eng. *eXtensible Markup Language*) predstavlja proširivi jezik za označavanje (eng. *markup*) tekstualnih dokumenata. Ideja je bila da se stvori jezik koji će i ljudi i računarski programi moći jednostavno da čitaju. `XML` definiše opštu sintaksu za označavanje podataka pomoću odgovarajućih etiketa (eng. *tags*) koje imaju poznato ili lako razumljivo značenje.

Disease. Tokom parsiranja baze `Orphanet Xref` postoji još dodatnih ograničenja. Entiteti koji nemaju vrednost za atribut `DisorderFlag` smatraju se validnim entitetima i biće sačuvani nakon parsiranja. Ukoliko se desi da neki od entiteta sadrži vrednost za atribut `DisorderFlag`, sačuvaće se samo entiteti čija je vrednost za taj atribut `head of classification`. Vrednost `head of classification` označava da se radi o najvišem nivou klasifikacije bolesti (npr. retke bolesti srca za `Orphanet` klasifikaciju retkih srčanih bolesti) [14].

Prilikom parsiranja `Xref` vrednosti baze `Orphanet Xref` bitno je napomenuti da postoje različite vrste relacija između entiteta baze `Orphanet Xref` i bolesti iz eksternih resursa na koju pokazuje `Xref` veza. Vrednost atributa `DisorderMappingRelation` ukazuje na to kakva je relacija [14]. Tipovi relacije mogu biti:

- `E` (eng. *Exact mapping*) - označava da je relacija identična
- `NTBT` (eng. *Narrower Term maps to a Broader Term*) - označava da se `Xref` vrednost preslikava na širi pojam od onog koji je odgovarajući trenutnom entitetu.
- `BTNT` (eng. *Broader Term maps to a Narrower Term*) - označava da se `Xref` vrednost preslikava na uži pojam od onog koji je odgovarajući trenutnom entitetu (u ovom slučaju je bitan `is_a` atribut baza `OBO` i `RGD` koji je opisan u delu 2.2.3, jer daje generalizaciju imena bolesti na koju pokazuje atribut `Xref`).

4.3.3 Parsiranje bibliotekom pronto

Biblioteka `pronto` predstavlja samostalnu biblioteku programskog jezika `Python` za raščlanjivanje, pregledanje, kreiranje i izvoz ontologija koja podržava nekoliko jezika i formata ontologije. Implementira specifikacije formata `OBO` u obliku bezbednog interfejsa visokog nivoa.

Baze koje koriste biblioteku `pronto` prilikom parsiranja su `OBO` i `RGD` i sačuvane su u formatu `.obo`. Parsiranje bibliotekom `pronto` započinje kreiranjem objekta klase `Ontology`. Klasa `Ontology` prilikom kreiranja objekta zahteva putanju do datoteke na disku, a kao opcioni argument moguće je podesiti broj niti koje će paralelno vršiti parsiranje datoteke. Korišćenje ove biblioteke prikazano je u kodu 4.5 na primeru parsiranja baza `OBO` i `RGD`.

4.3.4 Nekonvencionalno parsiranje

Baze koje zahtevaju nekonvencionalan pristup parsiranju su `HumsaVar`, `HPO` i `UniProt`. Ekstenzije datoteka ovih baza su `.txt` i `.dat` što može ukazivati na to da postoji specifičan format samo za te baze i da se ne mogu koristiti neke od biblioteka za parsiranje.

Prilikom čitanja baze `HumsaVar`, redovi koji će biti uvršteni u anotacijsku datoteku kao vrednost kolone `Variant category` treba da imaju `LP/P` (eng. *likely pathogenic or pathogenic*). Tokom parsiranja baze `UniProt`, moguće je da jedan entitet ima nekoliko vrednosti za atribut `Ensembl` ili `GeneID`. Kod takvih entiteta atribut koji ima nekoliko vrednosti biće sačuvan kao vrednost `None`. Spomenuti problem bi mogao da se reši odabirom neke od tih vrednosti ali ne postoji jednoznačan način da se izabere odgovarajuća vrednost tih atributa. Primer nekonvencionalnog parsiranja dat je u kodu 4.6 na primeru parsiranja baze `HPO`.

4.4 Pretprocesiranje uskladištenih podataka

Bitna faza procesa kreiranja anotacijske datoteke predstavlja pretprocesiranje podataka i inicijalizaciju klasa potrebnih za sam proces. Klasa `AnnotationContext` zadužena je za pretprocesiranje podataka i inicijalizaciju pomoćnih klasa. Ova klasa sadrži klasu zaduženu za skladištenje podataka `DBContext`, klasu `SearchEngineClient` zaduženu za komunikaciju i inicijalizaciju pretraživača, kao i klase zadužene za dohvatanje nedostajućih vrednosti atributa anotacijske datoteke (`class Symbol`, `class EntrezID`...). Ove klase ćemo u daljem tekstu nazivati atributskim klasama.

Pročitani podaci se nakon faze parsiranja skladište unutar klase `DBContext`. Ova klasa koristi se kao mesto odakle se može pristupiti sirovim pročitanim podacima svih baza. Različite metode ove klase omogućavaju jednostavnu interakciju sa podacima, gde je najkorišćeniji metod `GetDatabaseBySource`. Ovaj metod kao argument prima ime baze i kao povratnu vrednost ima strukturu `OrderedSet` koja sadrži parsirane podatke u vidu anotacijskog modela te baze. Primer dela ove klase prikazan je u kodu 4.7.

Uskladišteni podaci broje nekoliko stotina hiljada entiteta. Zbog ogromnog broja podataka sam proces parsiranja i kreiranja anotacijske datoteke bio bi vremenski zahtevan. Potrebno je koristiti efikasne strukture podataka koje mogu u što kraćem vremenskom periodu da daju poželjnu vrednost ili informaciju da vrednost ne po-

```
    @staticmethod
    def ReadOBO(returnObsoleteDOIDs=False, filePathOBO=OBO_PATH,
            filePathRGD=RGD_OBO_PATH):
        filePaths = [filePathRGD, filePathOBO]
        oboSet = OrderedSet()
        obsoleteSet = OrderedSet()

        for filePath in filePaths:
            oboData = Ontology(filePath,
                threads=multiprocessing.cpu_count())
            for term in oboData.terms():
                if term.obsolete:
                    doid = CheckEmpty(term.id)
                    if doid is not None:
                        obsoleteSet.add(doid)
                    continue

                doid = CheckEmpty(term.id)
                diseaseName = CheckEmpty(term.name)
                definition = CheckEmpty(term.definition)
                synonyms = term.synonyms
                parentDoids = list(term.superclasses(distance=1,
                    with_self=False).to_set())
                xrefs = term.xrefs
                altIds = term.alternate_ids
                oboSet.add(OBORow(doid, diseaseName, synonyms,
                    parentDoids, xrefs, altIds, definition))

      return (oboSet, obsoleteSet) if returnObsoleteDOIDs else oboSet
```

Listing 4.5: Statički metod za čitanje i parsiranje baza `OBO` i `RGD`

stoji. Rečnik[4] predstavlja strukturu koja je najviše zastupljena prilikom kreiranja anotacijske datoteke. Stoga, inicijalizacija rečnika predstavlja glavni fokus tokom ove faze kreiranja anotacijske datoteke. Dobro incijalizovani rečnici umnogome olakšaće finalnu fazu procesa kreiranja anotacijske datoteke.

[4]Rečnik je jedna od važnijih struktura podataka koja se obično koristi za skladištenje podataka u formatu ključ/vrednost. Svaki element u strukturi podataka obavezno ima ključ i neka vrednost je povezana sa tim ključem. Ključevi u rečniku moraju biti jedinstveni, dok vrednosti ne moraju.

```
    @staticmethod
    def ReadHPO(filePath=HPO_PATH):
        hpoSet = OrderedSet()
        with open(filePath, 'r') as hpoFile:
            hpoLines = hpoFile.readlines()
            for line in hpoLines[1:]:
                splittedLine = line.strip().split('\t')
                symbol = CheckEmpty(splittedLine[1])
                entrezID = CheckEmpty(splittedLine[0])
                omim = None
                orpha = None
                omimOrpha = CheckEmpty(splittedLine[8])
                if omimOrpha is not None:
                    omimOrphaSplitted = omimOrpha.split(":")
                    if omimOrphaSplitted[0] == "OMIM":
                        omim = omimOrphaSplitted[1]
                    elif omimOrphaSplitted[0] == "ORPHA":
                        orpha = omimOrphaSplitted[1]

                hpoSet.add(HPORow(symbol, entrezID, omim, orpha))

        return hpoSet
```

Listing 4.6: Statički metod za čitanje i parsiranje baze `HPO`

4.4.1 Metode pretprocesiranja

Klasa `AnnotationContext` sadrži tri bitne metode zadužene za pripremu podataka za narednu fazu `__InitializeDictionaries`, `__InitializeSearchEngineClient` i `__InitializeAttributes`. Ove metode biće opisane u nastavku kao zasebne celine.

4.4.1.1 Metod `__InitializeDictionaries`

Predstavlja metod zadužen za inicijalizaciju rečnika. Sama metoda je implementirana na način da se prvo dohvate podaci svih baza pomoću objekta klase `DBContext`. Prolaskom kroz podatke kreiraju se rečnici za svaki od atributa anotacijske datoteke, a pritom se formiraju i rečnici koji pomažu prilikom inicijalizacije nekih drugih elemenata korisnih za sam proces. Inicijalizovani rečnici koji se kori-

ste prilikom dohvatanja vrednosti atributa se prosleđuju dalje atributskim klasama pomoću metode `__InitializeAttributes` koja će biti opisana u nastavku, kao i same atributske klase. Deo metode je prikazan u kodu 4.8 na primeru inicijalizovanja rečnika koji se koriste u atributskoj klasi `Symbol`.

```
class DBContext:
    def __init__(self):
        self.__disGeNet = SourceInput.ReadDisGeNet()
        self.__cosmic = SourceInput.ReadCosmic()
        self.__clinvar = SourceInput.ReadClinVar()
            ...
        self.__obo, obsoleteDOIDs =
            MappingInput.ReadOBO(returnObsoleteDOIDs=True)
        self.__diseases = SourceInput.ReadDiseases(obsoleteDOIDs)

    def GetDatabaseBySource(self, source):
        if source is Source.DISGENET:
            return self.__disGeNet
        elif source is Source.COSMIC:
            return self.__cosmic
            ...
        elif source is Source.ORPHANET_XREF:
            return self.__orphanetXref
        elif source is Source.ENSEMBL:
            return self.__ensembl

        return None
```

Listing 4.7: Deo klase `DBContext` zadužene za skladištenje podataka

Neformalna podela rečnika koji se koriste prilikom kreiranja anotacijske datoteke bila bi na genske rečnike i rečnike koji se koriste za atributske klase vezane za bolest. Kao što je već poznato, genski deo veze sastoji se iz identifikatora koji se odlikuju svojom jedinstvenošću, što predstavlja pogodne vrednosti za rečnik. Genski rečnici predstavljaju veze između različitih atributa genskog dela veze. Na primer, `__entrezIDToSymbol` predstavlja rečnik koji kao ključ ima vrednost atributa `Entrez ID`, a kao vrednost atribut `Gene Symbol`.

Druga vrsta rečnika koja se odnosi na deo asocijacije koja je vezana za bolest zahteva komplikovaniji način uparivanja podataka. Ime bolesti predstavlja ni-

sku koja nije jedinstvena. Samim tim, prilikom kreiranja rečnika koji kao vrednost ključa treba da sadrži ime bolesti potrebno je dodatno pretprocesiranje imena bolesti. Jedan od načina pretraživanja atributa `DOID` zasniva se na rečniku `__diseaseNameFrozenSetToDOID` koji predstavlja vezu između pretprocesiranog imena bolesti zapakovanog kao `frozenset`[5] i atributa `DOID`. Da bi ime bolesti moglo da se koristi kao vrednost ključa u rečniku potrebno je da se pripremi na taj način da bude nepromenljivo, koncizno i da nakon procesa pripreme nosi identičnu informaciju o bolesti kao i pre. Način pripreme niske imena bolesti za vrednost ključa sastoji se iz pretprocesiranja niske i kreiranja `frozenset`-a. `Frozenset` svojim korišćenjem daje doprinos nepromenljivosti imena bolesti zbog nepromenljivosti (eng. *immutability*) što predstavlja glavnu osobinu ove strukture. Pretprocesiranje niske imena bolesti vrši se korišćenjem pomoćne funkcije `PreprocessingDiseaseName` koja je prikazana u kodu 4.9 i sam proces se sastoji iz sledećih koraka:

1. Crtice (-) se zamenjuju razmakom.
2. Kosa crta (/) se zamenjuje razmakom.
3. Nekoliko uzastopnih pojavljivanja belina zamenjuje se razmakom.
4. Znakovi interpunkcije se izbacuju iz imena bolesti.
5. Vrši se pretvaranje velikih u mala slova.
6. Vrši se tokenizacija[6].
7. Eliminisanje stop reči[7] (eng. *stopwords*).

   Koristi se lista engleskih reči u koju su dodate reči *Group* i *Type* koje predstavljaju reči koje se pojavljuju na velikom broju mesta, a nemaju značaj prilikom pretrage bolesti. Obrisana su slova *I*, *D*, *A* jer postoje bolesti koje koriste ova slova za označavanje tipa same bolesti i obrisane su reči *Down* i *With* jer se nalaze unutar liste stop reči, a korisne su prilikom pretrage. Na primer *Down syndrome* bi bio samo *syndrome*.

[5] `Frozenet` predstavlja strukturu podataka koja je nalik skupu, s tim što nakon inicijalizacije nije moguće modifikovati sam skup niti vrednosti koje sadrži.

[6] Proces tokenizacije predstavlja podelu teksta na tokene - elementarne jedinice sadržaja.

[7] Stop reči predstavljaju listu reči koje same po sebi ne nose informaciju.

8. Stemovanje[8] korišćenjem `ProterStemmer`-a.

9. Lematizacija[9] korišćenjem `WordNetLemmatizer`-a.

Nakon pretprocesiranja kao rezultat dobija se lista reči dobijena od imena bolesti. Na primer za bolest *breast-ovarian cancer, familial, susceptibility to, 4* kao rezultat pretprocesiranja dobiće se lista reči `[breast, ovarian, cancer, famili, suscept, 4]`. Ovako dobijenu listu reči i dalje je moguće modifikovati, pa je stoga nemoguće koristiti je kao vrednost ključa. Nakon kreiranja `frozenset`-a od ove liste dobija se novi objekat `frozenset({ovarian, suscept, breast, cancer, famili, 4})` koji više nije moguće modifikovati i kao takav pogodan je da se koristi kao vrednost ključa u rečniku.

4.4.1.2 Metod __InitializeSearchEngineClient

Kao što je već pomenuto u delu 4.2, unutar aplikacije `GDA` integrisan je program otvorenog koda `Typesense` koji se koristi prilikom pretraživanja teksta. Metod `__InitializeSearchEngineClient` zadužen je za inicijalizaciju pretraživača, a sam pretraživač je opisan klasom `SearchEngineClient`. Nakon detaljnog opisa klase `SearchEngineClient`, koja je bitna za sam proces inicijalizacije pretraživača jer se inicijalizacija potpuno oslanja na metode te klase, biće dalje opisana metoda `__InitializeSearchEngineClient`.

Klasa SearchEngineClient

Klasa `SearchEngineClient` predstavlja omotač oko klijentske biblioteke razvijene od strane `Typesense`-a i inicijalizuje se prilikom kreiranja objekta klase `AnnotationContext`. Konstruktor klase sadrži jedan obavezan i tri opciona parametra koji se mogu proslediti kao argumenti konstruktora. Obavezno je proslediti ime hosta (eng. *hostname*), dok opcioni parametri predstavljaju neka finija podešavanja samog pretraživača. Kao opcioni parametar može se proslediti:

[8]Stemovanje je jedan od pristupa za smanjenje varijabiliteta reči izvučenih iz nekog teksta. Koristi heuristiku i statistička pravila za odsecanje krajeva reči (tj. poslednjih nekoliko karaktera), gotovo bez razmatranja lingvističkih karakteristika reči. Na primer *argue, argued, argues, arguing -> argu.*

[9]Lematizacija je jedan od pristupa za smanjenje varijabiliteta reči izvučenih iz nekog teksta, kroz svođenje reči na njihov osnovni/koreni oblik. Koristi morfološki rečnik i primenjuje morfološku analizu reči, kako bi svela reč na njen osnovni oblik (koren reči definisan rečnikom) koji se naziva lema. Na primer *argue, argued, argues, arguing -> argue.*

```
# Symbol
if term.symbol is not None:
    symbol = PreprocessAttribute(term.symbol)
    # Symbol -> EntrezID
    if symbol not in self.__symbolToEntrezID
            and term.entrezID is not None:
        self.__symbolToEntrezID[symbol] = term.entrezID
    # Symbol -> UniprotID
    if symbol not in self.__symbolToUniprotID
            and term.uniprotID is not None:
        self.__symbolToUniprotID[symbol] = term.uniprotID
    # Symbol -> EnsemblID
    if symbol not in self.__symbolToEnsemblID
            and term.ensemblID is not None:
        self.__symbolToEnsemblID[symbol] = term.ensemblID
```

Listing 4.8: Deo metoda `__InitializeDictionaries`

- `API Key` - predstavlja ključ pomoću kog će klijent i server komunicirati.
- `Nodes` - predstavlja skup podešavanja u obliku rečnika i sadrži broj porta (eng. *port number*), protokol i ime hosta. Ako se unutar ovog parametra nalazi ime hosta, obavezni parametar imena hosta će biti zanemaren.
- `Connection Timeout Seconds` - predstavlja broj sekundi nakon kojih će se raskinuti veza klijent-server ako nema nikakve komunikacije između njih.

Klasa `SearchEngineClient` sadrži različite metode koje pružaju dodatne mogućnosti pri radu sa klijentskom stranom pretraživača, a to su:

- `CreateCollection(name, fields)` - predstavlja metod kojim se kreira šema kolekcije[10], dok se popunjavanje podacima vrši u zasebnoj metodi. Argumenti `name` i `fields` predstavljaju ime kolekcije i polja kolekcije respektivno.
- `DeleteCollection(name)` - predstavlja metod za brisanje kolekcije. Kolekcija se pronalazi na osnovu argumenta `name` koji predstavlja ime kolekcije.
- `GetAllCollectionNames` - predstavlja metod za dohvatanje imena svih dostupnih kolekcija.

[10]Kolekcija predstavlja skup podataka koji se mogu indeksirati, skladištiti i pretraživati.

- `ImportDataFromFile(name, filePath, batchSize=200)` - predstavlja metod za popunjavanje kolekcije `name` podacima sa diska na putanji `filePath`. Učitavanje podataka se vrši u delovima gde argument `batchSize` predstavlja broj redova koji će biti učitani kao celina.

- `SearchByQuery(name, query, queryBy, sortBy="_text_match:desc")` - predstavlja metod pomoću kog se pretražuje kolekcija `name`. Argumenti `query` i `queryBy` predstavljaju upit koji se pretražuje unutar kolekcije i polja po kojima se pretražuje respektivno. Ukoliko postoji nekoliko rezultata pretrage pomoću argumenta `sortBy` se određuje kako će biti sortirani.

Metod __InitializeSearchEngineClient

Metod `__InitializeSearchEngineClient` sadrži jedan argument `createCollection` koji predstavlja `boolean` vrednost i koji nosi informaciju o tome da li treba kreirati kolekciju. Na samom početku metode dohvataju se sva imena kolekcija pomoću metode `GetAllCollectionNames`. Prolaskom kroz listu dobijenih imena proverava se da li među njima postoji ime kolekcije koja se koristi prilikom kreiranja anotacijske datoteke. Ukoliko je kolekcija pronađena i argument `createCollection` sadrži vrednost `True`, pronađena kolekcija će biti obrisana pomoću metode `DeleteCollection`.

Kolekciju nije moguće dopuniti ili izmeniti i zbog toga jedini način ažuriranja kolekcije predstavlja brisanje stare i kreiranje nove. Nova kolekcija se kreira pomoću metode `CreateCollection`, šema kolekcije koja će biti kreirana za potrebe procesa kreiranja anotacijske datoteke sadrži tri polja `diseaseName`, `definition` i `DOID`. Kreiranje se vrši samo u slučaju da nije pronađena kolekcija sa istim imenom kao kolekcija koja se koristi prilikom kreiranja anotacijske datoteke ili da argument `createCollection` sadrži vrednost `True` tj. da je eksplicitno zatraženo da je potrebno kreirati novu kolekciju. Na taj način je osigurano da neće doći do kreiranja kolekcije ako je pre toga kreirana i pritom nije eksplicitno zatraženo kreiranje.

Nakon kreirane šeme kolekcije podaci se učitavaju koristeći skup `__searchEngineSet` koji je inicijalizovan u prethodno opisanoj metodi `__InitializeDictionaries`. Ovaj skup predstavlja uređenu trojku u kojoj se nalazi pretprocesirano ime bolesti, definicija imena bolesti i `DOID`. Prolaskom kroz ovaj skup, vrši se pretvaranje elemenata iz tročlane strukture u format `JSON` i upisuje se unutar `JSONL`[11] datoteke.

[11]Format `JSONL` (eng. *JSON Lines*) predstavlja skup `JSON` entiteta gde je svaki zapisan u zasebnom redu.

```
def PreprocessingDiseaseName(diseaseName, withoutList=False):
    if diseaseName is None:
        return None

    diseaseNameWithoutDash = diseaseName.replace('-', ' ')
    diseaseNameWithoutSlash = diseaseNameWithoutDash.replace('/', ' ')
    diseaseNameWithoutMultipleSpaces =
        re.sub(' +', ' ', diseaseNameWithoutSlash)

    diseaseNameWithoutPunctuation =
        RemovePunctuation(diseaseNameWithoutMultipleSpaces)

    diseaseNameLower = diseaseNameWithoutPunctuation.lower()
    diseaseNameTokenized = word_tokenize(diseaseNameLower)
    stopwords = nltk.corpus.stopwords.words("english")
    stopwords.remove("with").remove("i").append("type")
    diseaseNameWithoutStopwords = [word
        for word in diseaseNameTokenized if word not in stopwords]

    if not diseaseNameWithoutStopwords:
        diseaseNameWithoutStopwords = diseaseNameTokenized

    porterStemmer = PorterStemmer()
    diseaseNameStemmed = [porterStemmer.stem(word)
        for word in diseaseNameWithoutStopwords]

    wordnetLemmatizer = WordNetLemmatizer()
    diseaseNameLemmatized = [wordnetLemmatizer.lemmatize(word)
        for word in diseaseNameStemmed]

    return ' '.join(diseaseNameLemmatized)
        if withoutList else diseaseNameLemmatized
```

Listing 4.9: Funkcija `PreprocessingDiseaseName` koja se koristi za pretprocesiranje imena bolesti

Upisivanje u `JSONL` datoteku vrši se iz razloga što klijentska biblioteka `Typesense`-a jedino na taj način omogućava učitavanje količinski velikih podataka u kolekciju. Metodom `ImportDataFromFile` vrši se učitavanje podataka iz prethodno kreirane datoteke.

4.4.1.3 Metod __InitializeAttributes

Klasa `AnnotationContext` prilikom inicijalizacije definiše objekte atributskih klasa koji sadrže vrednosti `None`. Ovi objekti ne mogu da se inicijalizuju dok svi rečnici ne budu pripremljeni. Metod `__InitializeAttributes` se poziva kada su svi potrebni rečnici inicijalizovani i spremni da budu prosleđeni atributskim klasama. Shodno tome, zaduženje ove metode je da definisanim objektima atributskih klasa dodeli prave vrednosti tj. da se izvrši inicijalizacija svake od atributskih klasa. Objekti koji će biti inicijalizovani su: `symbol`, `entrezID`, `uniprotID`, `ensemblID`, `doid`, `diseaseName` i `xrefs`.

4.4.2 Atributske klase

Atributske klase predstavljaju skup klasa zaduženih za pronalaženje vrednosti nedostajućih atributa anotacijske datoteke. Svako polje ili atribut anotacijske datoteke predstavljeni su klasom čija je uloga da pomoću različitih metoda pruži jednostavan način za pronalaženje vrednosti tog atributa. Atributske klase sastoje se od klasa koje se odnose na gen i klasa koje se odnose na bolest. Svaka od ovih klasa sadrži metode koji predstavljaju načine pomoću kojih se mogu dohvatiti nedostajuće vrednosti atributa. Svaka atributska klasa kao argument prima odgovarajuće rečnike koji se koriste prilikom pretrage vrednosti atributa. U nastavku će biti opisane atributske klase pojedinačno i biće prikazane metode koje se koriste prilikom dohvatanja vrednosti samih atributa. Primer atributske klase dat je u kodu 4.10.

4.4.2.1 Atributske klase koje se odnose na gen

Atributskim klasama koje se odnose na gen pripadaju klase `Symbol`, `UniprotID`, `EnsemblID` i `EntrezID`, u nastavku biće dat opis svake klase zasebno. Kao što je već pomenuto, rečnici predstavljaju glavni način dohvatanja vrednosti atributa. Prilikom pronalaženja atributa koji pripadaju genskom delu veze rečnici predstavljaju jedini način pronalaženja i dohvatanja ovih atributa. Genski deo veze se svodi na pronalaženje vrednosti atributa na osnovu preostalih atributa iz genskog dela veze. Genska atributska klasa prikazana je u kodu 4.10 na primeru klase `Symbol`.

class Symbol

```
class Symbol:
    def __init__(self, entrezIDDict, ensemblIDDict, uniprotIDDict):
        self.__entrezIDDict = entrezIDDict
        self.__ensemblIDDict = ensemblIDDict
        self.__uniprotIDDict = uniprotIDDict

    def GetByEntrezID(self, entrezID):
        return self.__entrezIDDict[entrezID]
            if entrezID in self.__entrezIDDict else None

    def GetByEnsemblID(self, ensemblID):
        return self.__ensemblIDDict[ensemblID]
            if ensemblID in self.__ensemblIDDict else None

    def GetByUniProtID(self, uniprotID):
        return self.__uniprotIDDict[uniprotID]
            if uniprotID in self.__uniprotIDDict else None
```

Listing 4.10: Atributska klasa `Symbol`

Klasa `Symbol` predstavlja atributsku klasu za pronalaženje vrednosti polja `Symbol` anotacijske datoteke. Atribut `Symbol` nalazi se u svim izvornim bazama i u dve pomoćne baze `UniProt` i `HUGO`. Samim tim, te baze predstavljaju mesta odakle će se dohvatati vrednost atributa `Symbol`. Metode koje se koriste prilikom dohvatanja vrednosti ovog atributa su:

- `GetByEntrezID` - dohvatanje se vrši pomoću atributa `Entrez ID`. Za dohvatanje se koristi rečnik `__entrezIDToSymbol` koji kao vrednost ključa sadrži vrednost atributa `Entrez ID`.
- `GetByEnsemblID` - dohvatanje se vrši pomoću atributa `Ensembl ID`. Za dohvatanje se koristi rečnik `__ensemblIDToSymbol` koji kao vrednost ključa sadrži vrednost atributa `Ensembl ID`.
- `GetByUniProtID` - dohvatanje se vrši pomoću atributa `UniProt ID`. Za dohvatanje se koristi rečnik `__uniprotIDToSymbol` koji kao vrednost ključa sadrži vrednost atributa `UniProt ID`.

class EntrezID

Klasa `EntrezID` predstavlja atributsku klasu za pronalaženje vrednosti polja `Entrez ID` anotacijske datoteke. Atribut `Entrez ID` nalazi se u bazama `ClinVar`, `COSMIC`, `DisGeNet`, `HPO`, `UniProt`, `HUGO` i `Ensembl`. Samim tim, te baze predstavljaju mesta odakle će se dohvatati vrednost atributa `Entrez ID`. Metode koje se koriste prilikom dohvatanja vrednosti ovog atributa su:

- `GetBySymbol` - dohvatanje se vrši pomoću atributa `Symbol`. Za dohvatanje se koristi rečnik `__symbolToEntrezID` koji kao vrednost ključa sadrži vrednost atributa `Symbol`.
- `GetByEnsemblID` - dohvatanje se vrši pomoću atributa `Ensembl ID`. Za dohvatanje se koristi rečnik `__ensemblIDToEntrezID` koji kao vrednost ključa sadrži vrednost atributa `Ensembl ID`.
- `GetByUniProtID` - dohvatanje se vrši pomoću atributa `UniProt ID`. Za dohvatanje se koristi rečnik `__uniprotIDToEntrezID` koji kao vrednost ključa sadrži vrednost atributa `UniProt ID`.
- `GetByEnsemblProteinID` - dohvatanje se vrši pomoću atributa `Ensembl Protein ID`. Za dohvatanje se koristi rečnik `__ensemblProteinIDToEntrezID` koji kao vrednost ključa sadrži vrednost atributa `Ensembl Protein ID`.

class UniProtID

Klasa `UniProtID` predstavlja atributsku klasu za pronalaženje vrednosti polja `UniProt ID` anotacijske datoteke. Atribut `UniProt ID` nalazi se u bazama `HumsaVar`, `Orphanet`, `UniProt`, `HUGO` i `Ensembl`. Samim tim, te baze predstavljaju mesta odakle će se dohvatati vrednost atributa `UniProt ID`. Metode koje se koriste prilikom dohvatanja vrednosti ovog atributa su:

- `GetBySymbol` - dohvatanje se vrši pomoću atributa `Symbol`. Za dohvatanje se koristi rečnik `__symbolToUniprotID` koji kao vrednost ključa sadrži vrednost atributa `Symbol`.
- `GetByEntrezID` - dohvatanje se vrši pomoću atributa `Entrez ID`. Za dohvatanje se koristi rečnik `__entrezIDToUniprotID` koji kao vrednost ključa sadrži vrednost atributa `Entrez ID`.

- `GetByEnsemblID` - dohvatanje se vrši pomoću atributa `Ensembl ID`. Za dohvatanje se koristi rečnik `__ensemblIDToUniprotID` koji kao vrednost ključa sadrži vrednost atributa `Ensembl ID`.
- `GetByEnsemblProteinID` - dohvatanje se vrši pomoću atributa `Ensembl Protein ID`. Za dohvatanje se koristi rečnik `__ensemblProteinIDToUniprotID` koji kao vrednost ključa sadrži vrednost atributa `Ensembl Protein ID`.

class EnsemblID

Klasa `EnsemblID` predstavlja atributsku klasu za pronalaženje vrednosti polja `Ensembl ID` anotacijske datoteke. Atribut `Ensembl ID` nalazi se u bazama `Orphanet`, `UniProt`, `HUGO` i `Ensembl`. Samim tim, te baze predstavljaju mesta odakle će se dohvatati vrednost atributa `Ensembl ID`. Metode koje se koriste prilikom dohvatanja vrednosti ovog atributa su:

- `GetBySymbol` - dohvatanje se vrši pomoću atributa `Symbol`. Za dohvatanje se koristi rečnik `__symbolToEnsemblID` koji kao vrednost ključa sadrži vrednost atributa `Symbol`.
- `GetByEntrezID` - dohvatanje se vrši pomoću atributa `Entrez ID`. Za dohvatanje se koristi rečnik `__entrezIDToEnsemblID` koji kao vrednost ključa sadrži vrednost atributa `Entrez ID`.
- `GetByUniProtID` - dohvatanje se vrši pomoću atributa `UniProt ID`. Za dohvatanje se koristi rečnik `__uniprotIDToEnsemblID` koji kao vrednost ključa sadrži vrednost atributa `UniProt ID`.
- `GetByEnsemblProteinID` - dohvatanje se vrši pomoću atributa `Ensembl Protein ID`. Za dohvatanje se koristi rečnik `__ensemblProteinIDToEnsemblID` koji kao vrednost ključa sadrži vrednost atributa `Ensembl Protein ID`.

4.4.2.2 Atributske klase koje se odnose na bolest

Atributskim klasama koje se odnose na bolest pripadaju klase `DOID`, `DiseaseName` i`Xrefs`, u nastavku biće dat opis svake klase zasebno. Ove klase zbog strukture atributa koje predstavljaju osim rečnika koriste i druge načine pronalaženja i dohvatanja vrednosti atributa. Klasa `Xrefs` ne odnosi se direktno na bolest ali koristi se prilikom

dohvatanja atributa DOID, pa će zato biti opisana zajedno sa atributskim klasama koje se odnose na bolest.

class DOID

Klasa DOID predstavlja atributsku klasu za pronalaženje vrednosti polja DOID anotacijske datoteke. Atribut DOID nalazi se u bazama Diseases, OBO i RGD. Samim tim, te baze predstavljaju mesta odakle će se dohvatati vrednost atributa DOID. Metode koje se koriste prilikom dohvatanja vrednosti ovog atributa su:

- GetByOmim - dohvatanje se vrši pomoću Xref vrednosti OMIM. Za dohvatanje se koristi rečnik __omimToDOID koji kao vrednost ključa sadrži Xref vrednost OMIM.
- GetByUmls - dohvatanje se vrši pomoću Xref vrednosti UMLS. Za dohvatanje se koristi rečnik __umlsToDOID koji kao vrednost ključa sadrži Xref vrednost UMLS.
- GetByGard - dohvatanje se vrši pomoću Xref vrednosti GARD. Za dohvatanje se koristi rečnik __gardToDOID koji kao vrednost ključa sadrži Xref vrednost GARD.
- GetByMesh - dohvatanje se vrši pomoću Xref vrednosti MeSH. Za dohvatanje se koristi rečnik __meshToDOID koji kao vrednost ključa sadrži Xref vrednost MeSH.
- GetByMedDra - dohvatanje se vrši pomoću Xref vrednosti MedDRA. Za dohvatanje se koristi rečnik __medDraToDOID koji kao vrednost ključa sadrži Xref vrednost MedDRA.
- GetByIcd10 - dohvatanje se vrši pomoću Xref vrednosti ICD-10. Za dohvatanje se koristi rečnik __icd10ToDOID koji kao vrednost ključa sadrži Xref vrednost ICD-10.
- GetByXref - predstavlja objedinjeni način dohvatanja pomoću Xref vrednosti. Kao argument se šalje tip Xref vrednosti i sama vrednost. Dohvatanje se dalje propagira određenoj metodi na koju se odnosi prosleđeni Xref tip.
- GetByDiseaseName - dohvatanje se vrši pomoću atributa Disease Name. Za dohvatanje se koristi rečnik __diseaseNameFrozenSetToDOID koji kao vrednost ključa sadrži frozenset sastavljen od pretprocesiranog imena bolesti.

Ovaj metod kao argument prima ime bolesti, pa zatim vrši pretprocesiranje imena bolesti i nakon toga se od rezultata pretprocesiranja kreira struktura `frozenset` pomoću koje se pretražuje rečnik.

- `GetByDiseaseNameUsingSearchEngine` - dohvatanje se vrši pomoću atributa `Disease Name`. Za dohvatanje se koristi integrisani pretraživač `Typesense` i rečnik `__diseaseNameFrozenSetToDOID` koji kao vrednost ključa sadrži `frozenset` sastavljen od pretprocesiranog imena bolesti. Ovaj metod kao argument prima ime bolesti, pretraga se prvo vrši po istom principu kao i u prethodnoj metodi `GetByDiseaseName` pomoću rečnika. Ukoliko sam rečnik ne sadrži traženu vrednost, pretraga se nastavlja korišćenjem pretraživača.

  Metoda `__SearchWithSearchEngine` koristi se prilikom korišćenja pretraživača i prikazana je u kodu 4.12. Ova metoda vrši pretraživanje kolekcije koja se koristi za kreiranje anotacijske datoteke. Argument ove metode je ime bolesti zadato kao lista tokena, koja se zatim spaja u nisku gde su tokeni razdvojeni razmakom. Sama atributska klasa prilikom inicijalizacije kao argument konstruktora prima objekat klase `SearchEngineClient` koji predstavlja klijenta za komunikaciju sa pretraživačem. Pomoću metode tog objekta `SearchByQuery` vrši se pretraga. Parametri koji se šalju su ime kolekcije u kojoj se vrši pretraga, pretprocesirano ime bolesti što predstavlja sam upit koji se pretražuje i polja po kojima će biti vršena pretraga. Pretraga se vrši pomoću dva polja kolekcije, a to su `diseaseName` i `definition`. Pretraga pomoću polja `diseaseName` predstavlja osnovu koja ne zahteva dodatna pojašnjenja zašto se koristi baš to polje za pretragu. Polje `definition` predstavlja nadogradnju koja doprinosi preciznošću same pretrage. Ovo polje predstavlja definiciju imena bolesti i sadrži različite termine koji se nekad mogu naći u upitu, a nema ih u samom imenu bolesti, pa su kao takvi jako značajni za pretragu i povećavaju preciznost pretrage u slučajevima kada ime bolesti može da ima nekoliko različitih termina.

  Ukoliko nije pronađen nijedan rezultat vrši se dodatna optimizacija upita. Dodatna optimizacija upita odnosi se na određene termine u imenu bolesti koji su ustaljeni i koje koristi većina imena. Takvi termini se odnose na neke specifičnosti pri samoj bolesti. Na primer, bolest *dehydrated hereditary stomatocytosis 1 with or without pseudohyperkalemia and/or perinatal edema* sadrži specifičnost koja je opisana u nastavku bolesti. Takve specifičnosti mogu da otežaju pretragu i da dovedu do toga da `DOID` uopšte ne bude pronađen. Op-

timizacija se odnosi na to da ako upit sadrži termine kao što su `due`, `with without`, `with` ili `without` pretraga će se izvršiti ponovno samo sa prvim delom bolesti koji se nalazi pre ovih termina. Na prethodno datom primeru novi upit bi sadržao samo nisku pre termina `with or without` tj. *dehydrated hereditary stomatocytosis 1*. Treba napomenuti da bi se celokupna optimizacija vršila nad već pretprocesiranim imenom bolesti što je u ovom primeru preskočeno. Primer ove metode dat je u kodu 4.11.

```
def GetByDiseaseNameUsingSearchEngine(self, diseaseName):
   if diseaseName is None:
     return None, None

   preprocessedDiseaseName = PreprocessingDiseaseName(diseaseName)
   preprocessedDiseaseNameFrozenSet =
        frozenset(preprocessedDiseaseName)
   if preprocessedDiseaseNameFrozenSet in
        self.__diseaseNameFrozenSetDict:
     return self.__diseaseNameFrozenSetDict[
        preprocessedDiseaseNameFrozenSet], DOID_SOURCE_FROZEN_SET
   else:
     return self.__SearchWithSearchEngine(preprocessedDiseaseName)
```

Listing 4.11: Metoda `GetByDiseaseNameUsingSearchEngine` atributske klase `DOID`

class DiseaseName

Klasa `DiseaseName` predstavlja atributsku klasu za pronalaženje vrednosti polja `Disease Name` anotacijske datoteke. Atribut `Disease Name` nalazi se u bazama `ClinVar`, `COSMIC`, `Diseases`, `DisGeNet`, `HumsaVar`, `Orphanet`, `Orphanet Xref`, `RGD` i `OBO`. Samim tim, te baze predstavljaju mesta odakle će se dohvatati vrednost atributa `Disease Name`. Metode koje se koriste prilikom dohvatanja vrednosti ovog atributa su:

- `GetByDoid` - dohvatanje se vrši pomoću atributa `DOID`. Za dohvatanje se koristi rečnik `__doidToDiseaseName` koji kao vrednost ključa sadrži vrednost atributa `DOID`.

```
    def __SearchWithSearchEngine(self, preprocessedDiseaseNameTokens):
        preprocessedDiseaseName =
            ' '.join(preprocessedDiseaseNameTokens)
        searchResult = self.__searchEngineClient.SearchByQuery(
                                            COLLECTION_NAME_DOID,
                                            preprocessedDiseaseName,
                                            QUERY_BY_DOID)
        if len(searchResult["hits"]) > 0:
            foundDiseaseName =
                searchResult["hits"][0]["document"]["diseaseName"]
            jaccSimilarity = JaccardSimilarity(foundDiseaseName,
                                            preprocessedDiseaseName)
            doid = searchResult["hits"][0]["document"]["doid"]
            doidSource = DOID_SOURCE_SEARCH_ENGINE + ", " +
                str(int(round(jaccSimilarity * 100, 0))) + '%'

            return doid, doidSource
        elif " due " in preprocessedDiseaseName:
            return self.GetByDiseaseNameUsingSearchEngine(
                        preprocessedDiseaseName.split(" due ")[0])
        elif " with without " in preprocessedDiseaseName:
            return self.GetByDiseaseNameUsingSearchEngine(
                    preprocessedDiseaseName.split(" with without ")[0])
        elif " with " in preprocessedDiseaseName:
            return self.GetByDiseaseNameUsingSearchEngine(
                        preprocessedDiseaseName.split(" with ")[0])
        elif " without " in preprocessedDiseaseName:
            return self.GetByDiseaseNameUsingSearchEngine(
                        preprocessedDiseaseName.split(" without ")[0])

        return None, None
```

Listing 4.12: Pomoćna metoda `__SearchWithSearchEngine` atributske klase `DOID`

- `GetByOrpha` - dohvatanje se vrši pomoću `ORPHACode`-a. Za dohvatanje se koristi rečnik `__orphaToDiseaseName` koji kao vrednost ključa sadrži vrednost `ORPHACode`.
- `GetByOmim` - dohvatanje se vrši pomoću `Xref` vrednosti `OMIM`. Za dohvatanje se koristi rečnik `__omimToDiseaseName` koji kao vrednost ključa sadrži `Xref`

vrednost OMIM. Sam metod može da pronađe više od jedne bolesti, pa je stoga povratna vrednost lista pronađenih bolesti.

- GetByOmimDoidAndDiseaseName - predstavlja metod koji pruža dohvatanje para DOID - Disease Name, ukoliko neka od baza ne sadrži nijedan od ova dva atributa. Dohvatanje se vrši pomoću Xref vrednosti OMIM. Za dohvatanje se koristi rečnik __omimToDoidAndDiseaseName koji kao vrednost ključa sadrži Xref vrednost OMIM. Sam metod može da pronađe više od jednog para DOID - Disease Name, pa je stoga povratna vrednost lista pronađenih parova. Baza HPO koristi ovaj metod prilikom finalnog parsiranja.

- GetParentDoidAndDiseaseNamesByDoid - predstavlja metod koji pruža dohvatanje parent Disease Name-a i njihovih DOID-a. Dohvatanje se vrši pomoću atributa DOID. Rečnik __doidToParentDoidAndDiseaseName koristi se za dohvatanje koji kao vrednost ključa sadrži vrednost atributa DOID. Sam metod može da pronađe više od jednog para parent Disease Name - DOID, pa je stoga povratna vrednost lista pronađenih parova.

class Xrefs

Klasa Xrefs predstavlja atributsku klasu za dohvatanje Xref vrednosti na osnovu ORPHACode-a. Xref vrendnosti su sastavni deo baze Orphanet Xref. Samim tim, ta baza predstavlja mesto odakle će se dohvatati Xref vrednosti. Metode koje se koriste prilikom dohvatanja ovih vrednosti predstavljene su u četiri kategorije na koje su podeljene Xref vrednosti:

- GetByOrphaExact - dohvatanje se vrši pomoću ORPHACode-a. Za dohvatanje se koristi rečnik __orphaToExactXrefs koji kao vrednost ključa sadrži vrednost ORPHACode.

- GetByOrphaBtnt - dohvatanje se vrši pomoću ORPHACode-a. Za dohvatanje se koristi rečnik __orphaToBtntXrefs koji kao vrednost ključa sadrži vrednost ORPHACode.

- GetByOrphaNtbt - dohvatanje se vrši pomoću ORPHACode-a. Za dohvatanje se koristi rečnik __orphaToNtbtXrefs koji kao vrednost ključa sadrži vrednost ORPHACode.

- `GetByOrphaOther` - dohvatanje se vrši pomoću `ORPHACode`-a. Za dohvatanje se koristi rečnik `__orphaToOtherXrefs` koji kao vrednost ključa sadrži vrednost `ORPHACode`.

Ove kategorije se detaljnije mogu videti u delu 4.3.2, gde su opisane prve tri kategorije dok četvrta kategorija predstavlja preostale `Xref` vrednosti koje nisu pripale nijednoj od prve tri. Povratna vrednost ovih metoda jeste struktura rečnik koja kao vrednost ključeva ima tipove `Xref` vrednosti, a kao vrednosti sadrži strukturu skup u kojoj se nalaze `Xref` vrednosti. U primeru 4.13 može se videti povratna vrednost opisanih metoda u vidu strukture `OrderedDict`.

```
{
  '141': OrderedDict{
           {<Xref.UMLS: 1>: ['C0206307', 'C3542499']},
           {<Xref.MeSH: 2>: ['D017825']},
           {<Xref.GARD: 3>: ['5984']},
           {<Xref.MedDRA: 4>: ['10067608']},
           {<Xref.OMIM: 5>: ['271900']}
         }
}
```

Listing 4.13: Deo rečnika `__orphaToExactXrefs`

4.5 Pretraga nedostajućih atributa i kreiranje anotacijske datoteke

Do sada, svi koraci koji su prethodili predstavljali su pripremu za finalno parsiranje. Finalni deo kreiranja anotacijske datoteke predstavlja pretragu i dohvatanje nedostajućih vrednosti atributa i razvrstavanje tih vrednosti u celinu koja se naziva anotacijska datoteka. Klasa zadužena za dohvatanje i pretragu nedostajućih atributa prilikom finalnog parsiranja naziva se `ParsingContextThread`. Ova klasa sadrži objekte klasa `DBContext` i `AnnotationContext`, koji sadrže sve neophodne elemente za finalno parsiranje. Po nazivu klase može se videti da je klasa implementirana i razvijena na način da koristi niti prilikom izvršavanja. Korišćenje niti predstavlja pogodno rešenje koje se nameće za problem pretrage nedostajućih atributa prilikom finalnog parsiranja. Pored toga što će se time ubrzati proces pretrage i dohvatanja

nedostajućih vrednosti, problem će biti razložen na pretraživanje svake baze pojedinačno. Dakle, ideja je da se dohvatanje i pretraga nedostajućih vrednosti izvršava u zasebnoj niti za svaku bazu. Nakon što sve niti završe sa izvršavanjem rezultati će biti objedinjeni i predstavljeni kao anotacijska datoteka. Klasa `ParsingContextThread` sadrži nekoliko metoda koje se koriste prilikom parsiranja i biće opisane u nastavku.

4.5.1 Metod __ParseSources

Metod `__ParseSources` koristi se za pripremu paralelizacije i inicijalizaciju niti. Implementacija paralelizacije vrši se pomoću klase `ThreadPoolExecutor`, koja se koristi za kreiranje i upravljanje nitima. Kao što je već pomenuto za svaku bazu vrši se parsiranje u zasebnoj niti. Metoda koju svaka nit izvršava naziva se `__ParseSource` i biće opisana u nastavku. Nakon inicijalizacije i pokretanja izvršavanja niti, klasa `ThreadPoolExecutor` pruža mogućnost dohvatanja rezultata niti. Kao rezultat svaka nit šalje ime baze nad kojom je vršeno parsiranje i ishod parsiranja u vidu skupa koji je sačinjen od redova anotacijske datoteke. Za objedinjavanje rezultata koristi se rečnik `__sourcesAnnotationSetDict`. Ovaj rečnik kao vrednosti ključeva ima nazive baza, a kao vrednosti sadrži strukturu skup. Rezultat se skladišti unutar rečnika čija je namena da sakupi rezultate svih baza nakon što niti završe parsiranje. Primer ove metode dat je u kodu 4.14.

4.5.2 Metod __ParseSource

Metod `__ParseSource` koristi se za finalno parsiranje pojedinačnih baza i izvršava se u svakoj niti za različite baze. Kao argument ove metode prosleđuje se ime baze koja se parsira. Na samom početku inicijalizuje se prazan skup u kojem će biti uskladišteni redovi anotacijske datoteke i vrši se dohvatanje redova baze pomoću metode `GetDatabaseBySource` klase `DBContext`. Nakon toga vrši se prolazak kroz redove baze koja se parsira. Prilikom prolaska vrši se dohvatanje i pretraga nedostajućih vrednosti atributa same baze. Ideja je da se prilikom prolaska za svaki red baze pokuša dohvatanje što je više moguće nedostajućih vrednosti atributa. Dohvatanje i pretraga se vrši kroz četiri faze koje će biti opisane u nastavku.

```
    def __ParseSources(self, progress):
        with Progress(SpinnerColumn(),
            TextColumn("[progress.description]{task.description}"),
            BarColumn(), TaskProgressColumn(), MofNCompleteColumn(),
           TimeElapsedColumn(), TimeRemainingColumn()) as progressBar:

            with ThreadPoolExecutor() as executor:
                progressTask = progressBar.add_task("Parsing",
                    total=self.dbContext.GetTotalParsingLength())
                for source, sourceSet in executor.map(
                    lambda args: self.__ParseSource(*args),
                        [(source, progress, progressBar, progressTask)
                                    for source in self.__sources]):
                   self.__sourcesAnnotationSetDict[source] = sourceSet
```

Listing 4.14: Pomoćna metoda `__ParseSources` klase `ParsingContextThread`

- **Faza I - Inicijalizacija i priprema**

  Ova faza sastoji se iz inicijalizacije atributa, provere nedostajućih vrednosti atributa i pripreme pomoćnih metoda za dohvatanje nedostajućih vrednosti. Kao što je već rečeno, na samom početku vrši se inicijalizacija atributa tj. kreiraju se atributske promenljive koje dohvataju vrednosti atributa trenutnog reda u bazi koja se parsira. Ukoliko se parsira baza `Diseases`, atributskoj promenljivoj `doidSource` biće dodeljena vrednost *Database*.

  Provera nedostajućih vrednosti vrši se nad atributima gena, što je prikazano u kodu 4.15. Ova provera je bitna zarad popunjavanja niza `noneAttributes` oznakama atributa koji sadrže nedostajuće vrednosti. Ovaj niz se koristi kao optimizacija prilikom kreiranja redosleda pretraživanja atributa gena što će biti detaljnije opisano u sledećoj fazi. Pretraga i dohvatanje genskog dela veze zahteva dodatne elemente i pripreme radi efikasnije implementacije i bolje struktuiranosti izvornog koda.

  Prilikom dohvatanja atributa gena koriste se parcijalne funkcije[12] (eng. *partial functions*). Metod `partial` koristi se za kreiranje parcijalnih funkcija i deo je modula programskog jezika `Python` koji se naziva `functools`. Parcijalne funkcije koriste se da omoguće dinamičko izvršavanje metoda atributskih klasa u svrhu pretraživanja i dohvatanja vrednosti atributa. Za objedinjavanje parcijalnih funkcija koriste se

[12]Parcijalna funkcija omogućava da se funkcija koja ima n argumenata, pozove sa manje argumenata gde bi preostali argumenti bili fiksirani određenim vrednostima.

```
noneAttributes = []
if symbol is None:
    noneAttributes.append(Attribute.SYMBOL)
if entrezID is None:
    noneAttributes.append(Attribute.ENTREZ_ID)
if uniprotID is None:
    noneAttributes.append(Attribute.UNIPROT_ID)
if ensemblID is None:
    noneAttributes.append(Attribute.ENSEMBL_ID)
```

Listing 4.15: Deo provere nedostajućih vrednosti gena

dodatni ugnežđeni metodi za sva četiri atributa. Princip rada ovih metoda je sličan za sve atribute gena pa će biti opisan na primeru atributa `Symbol`.

Metod koji se koristi za objedinjavanje parcijalnih funkcija za genski atribut `Symbol` naziva se `partialGetMethodsSymbol`, implementiran je kao ugnežđeni metod unutar metode `__ParseSource` i prikazan u kodu 4.16. Ovaj metod kao argumente prima vrednosti preostalih genskih atributa, u slučaju atributa `Symbol` to su atributi `Entrez ID`, `UniProt ID` i `Ensembl ID`. Metod je implementiran na način da se za svaki prosleđeni argument proverava da li predstavlja nedostajuću vrednost. Ukoliko prosleđeni argument sadrži vrednost tj. ne predstavlja nedostajuću onda se može koristiti za pretragu samog atributa, u ovom slučaju atributa `Symbol`. Niz `partialMethods` koristi se kao povratna vrednost ove metode i objedinjuje parcijalne funkcije sastavljene od metoda atributskih klasa, u ovom slučaju od metoda atributske klase `Symbol`. U niz `partialMethods` dodaje se nova parcijalna funkcija kreirana od metode atributske klase na koju se odnosi argument za koji je proverena nedostajuća vrednost i sam taj argument se fiksira kao parametar parcijalne funkcije.

Preostali atributi koriste druge metode koji se nazivaju `partialGetMethodsEntrezID`, `partialGetMethodsUniprotID` i `partialGetMethodsEnsemblID` čija je implementacija identična metodi `partialGetMethodsSymbol` s tim što koriste drugačije atribute.

```
def partialGetMethodsSymbol(entrezIDP, uniprotIDP, ensemblIDP):
    partialMethods = []
    if entrezIDP is not None:
        partialMethods.append(partial(
            self.annotationContext.symbol.GetByEntrezID, entrezIDP))

    if uniprotIDP is not None:
        partialMethods.append(partial(
            self.annotationContext.symbol.GetByUniProtID, uniprotIDP))

    if ensemblIDP is not None:
        partialMethods.append(partial(
            self.annotationContext.symbol.GetByEnsemblID, ensemblIDP))

    return partialMethods
```

Listing 4.16: Ugnežđena metoda `partialGetMethodsSymbol`

- **Faza II - Pretraga i dohvatanje genskog dela veze**

  U ovoj fazi biće objašnjeno na koji način se pretražuju i dohvataju nedostajuće vrednosti atributa gena. Pod atributima gena podrazumevaju se `Symbol`, `Entrez ID`, `UniProt ID` i `Ensembl ID`.

  Pre nego što bude opisan univerzalan način dohvatanja vrednosti atributa bitno je spomenuti da ova faza sadrži specifičnost koja se odnosi na parsiranje baze `Diseases`. Ova baza opisana je u delu 2.1.7 i može se videti da je jedan od atributa identifikator proteina koji se može iskoristiti kao veza prema atributima `Entrez ID`, `UniProt ID` i `Ensembl ID`. Samim tim, ukoliko se vrši parsiranje redova ove baze prvo će se proveriti na osnovu identifikatora proteina da li je moguće dohvatiti neki od ova tri atributa.

  U prethodnoj fazi dat je opis procesa pripreme različitih elemenata koji utiču na efikasnost i neophodni su za realizaciju ove faze. Jedan od tih elemenata je i niz `noneAttributes` koji sadrži oznake atributa koji nemaju vrednost. Ovaj niz predstavlja početnu tačku za dalju pretragu nedostajućih vrednosti atributa. Ukoliko bi se samo prolaskom kroz ovaj niz vršila pretraga dolazi se do problema gubljenja vrednosti atributa. Taj problem se odnosi na to da ukoliko se unutar niza nedostajućih vrednosti nađe nekoliko atributa gde je dovoljno da jedan od njih svoju vrednost može da pronađe samo pomoću atributa koji su u nizu i da se nalazi pre

njih, ta vrednost će biti izgubljena. Ovaj problem se rešava korišćenjem funkcije `permutations` pomoću koje se pravi novi niz koji će sadržati permutacije bez ponavljanja sastavljene od elemenata niza `noneAttributes`. Bitnost ovog koraka je u tome što se pruža mogućnot da i atributi koji inicijalno ne sadrže vrednost, nakon dohvatanja njihove vrednosti budu iskorišćeni za pronalazak ostalih nedostajućih vrednosti atributa.

Dobijanjem niza koji sadrži sve moguće redoslede pretrage započinje se faza pretrage i dohvatanja vrednosti atributa. Prolaskom kroz ovaj niz svakom iteracijom dobija se novi redosled pretrage. Ne mora da znači da će uvek biti potrebno proći kroz sve redoslede pretrage što može biti vremenski zahtevo. Ukoliko se ispostavi da su svi atributi pronađeni pretraga će biti prekinuta, isto tako ukoliko se ispostavi da nijedan od atributa nema vrednost pretraga neće ni biti započeta. Sama pretraga se vrši tako što se prolazi kroz niz koji predstavlja redosled pretrage tj. taj niz sadrži tipove atributa za koje je potrebno pronaći vrednost.

```
    def GetAttribute(getMethods):
    for getMethod in getMethods:
        attribute = getMethod()
        if attribute is not None:
            return attribute

    return None
```

Listing 4.17: Pomoćna metoda `GetAttribute`

Pretraga se vrši korišćenjem pomoćne metode `GetAttribute` koja kao argument prima niz parcijalnih funkcija i prolaskom kroz taj niz izvršava ih i vraća prvu pronađenu vrednost, primer ove metode dat je u kodu 4.17. Niz parcijalnih funkcija koji se prosleđuje metodi `GetAttribute` kreira se pomoću metoda opisanih u prvoj fazi. Prikaz dela pretrage i dohvatanja dat je u kodu 4.18 na primeru dohvatanja vrednosti atributa `Symbol`.

```
    if symbol is not None or entrezID is not None or
            uniprotID is not None or ensemblID is not None:
        ordersOfSearch = list(permutations(noneAttributes))
        stopSearch = False
        for orderOfSearch in ordersOfSearch:
            for attribute in orderOfSearch:
                if symbol is None and attribute is Attribute.SYMBOL:
                    symbol = GetAttribute(
                                partialGetMethodsSymbol(
                                    PreprocessAttribute(entrezID),
                                    PreprocessAttribute(uniprotID),
                                    PreprocessAttribute(ensemblID)))

                if symbol is not None and entrezID is not None and
                      uniprotID is not None and ensemblID is not None:
                    stopSearch = True
                    break

            if stopSearch:
                break
```

Listing 4.18: Deo pretrage i dohvatanja atributa gena

- **Faza III - Pretraga i dohvatanje atributa vezanih za bolest**

  Do sada atributi koji su pronađeni odnose se na genski deo veze, u ovom delu biće prikazano i objašnjeno na koji način se pretražuju i dohvataju atributi vezani za bolest. `DOID` i `Disease Name` predstavljaju atribute koji se odnose na bolest. Ovi atributi zbog svoje specifičnosti i strukture sadrže veliki broj specijalnih slučajeva kada se vrši njihova pretraga. Pretraga tokom ove faze može se podeliti na nekoliko koraka:

  1. Prvi korak pretrage započinje specijalnim slučajem vezanim za bazu `HPO` koja je opisana u delu 2.1.6. Ova baza ne sadrži nijedan od atributa vezanih za bolest, ali sadrži vezu ka njima preko kodova `OMIM` i `ORPHA`. Ranije je spomenuto da kod `ORPHA` sadrži jedinstvenu vezu ka imenu bolesti, dok kod `OMIM` može dati nekoliko različitih imena bolesti. Redovi baze `HPO` mogu sadržati kod `ORPHA` ili `OMIM`, ali ne oba, pa samim tim način dohvatanja bolesti zavisi od toga koji od ova dva koda sadrži trenutni red koji se obrađuje.

     U ovom delu pretrage, korišćenjem koda `ORPHA` dohvatanje se vrši samo

za ime bolesti i realizuje se na jednostavan način pomoću metode `GetByOrpha` atributske klase `DiseaseName`, dok će dohvatanje atributa `DOID` biti realizovano kasnije.

Korišćenjem koda `OMIM` vrši se dohvatanje oba atributa i to pomoću metode `GetByOmimDoidAndDiseaseName` atributske klase `DiseaseName`. Ova metoda kao povratnu vrednost daje listu parova `Disease Name` - `DOID`, iz razloga što red baze `HPO` može sadržati nekoliko vrednosti vezanih za bolest dohvaćene na osnovu koda `OMIM` koje treba sačuvati u anotacijsku datoteku. Ukoliko se ustanovi da postoje elementi dobijene liste koji ne sadrže vrednost atributa `DOID`, a sadrže vrednost atributa `Disease Name`, pokušaće se pretraga vrednosti atributa `DOID` pomoću imena bolesti metodom `GetByDiseaseNameUsingSearchEngine` atributske klase `DiseaseName`. Na osnovu dobijene liste i pronađenih vrednosti atributa `DOID` i `Disease Name` kreira se niz `doidAndDiseaseNames` u kom će biti smešteni atributi `DOID`, `DOID Source` i `Disease Name` kao uređena trojka.

Fokus u narednim koracima biće na pretrazi i dohvatanju atributa `DOID` iz razloga što preostale baze sadrže ime bolesti.

2. Drugi korak pretrage i dohvatanja atributa `DOID` započinje specijalnim slučajem za baze `DisGeNet` i `ClinVar`, koje kao jedan od atributa sadrže `Xref` vrednost `UMLS`. Dohvatanje se vrši na jednostavan način metodom `GetByUmls` atributske klase `DOID` i prikazano je u primeru koda 4.19.

```
if (source is Source.DISGENET or source is Source.CLINVAR) and
    term.umls is not None and doid is None:
    doid, doidSource =
        self.annotationContext.doid.GetByUmls(term.umls)
```

Listing 4.19: Dohvatanja atributa `DOID` pomoću `Xref` vrednosti `UMLS`

Takođe, baze `HumsaVar` i `ClinVar` kao jedan od atributa sadrže `Xref` vrednost `OMIM`. Dohvatanje se vrši na jednostavan način metodom `GetByOmim` atributske klase `DOID`. Ukoliko ova metoda ne uspe da pronađe nijednu vrednost pokušava se i pretraga pomoću metode `GetByDiseaseNameUsingSearchEngine` iste atributske klase, gde se kao ime bolesti prosleđuje niska u formatu `"OMIM:<OMIM vredost>"`. Razlog ove pretrage jeste što baza `RGD` sadrži nekoliko entiteta koji kao vrednost atributa `synonym` sadrže vrednosti koda `OMIM` u prethodno opisanom formatu koji se prosleđuje kao niska koja se pretražuje.

3. Treći korak odnosi se na specijalizovane pretrage baza koje sadrže atribut `ORPHACode`, a to su `Orphanet` i `HPO`. Baza `Orphanet Xref` značajno utiče na mogućnost ove pretrage jer njen sadržaj pruža `Xref` vrednosti zajedno sa `ORPHACode`-ovima. Atributska klasa `Xrefs` koja sadrži `Xref` vrednosti koje su raspoređene u četiri kategorije koristiće se kao mesto odakle će biti dohvaćene potrebne `Xref` vrednosti koje se koriste prilikom pretrage, ova klasa opisana je u delu 4.4.2.2. Na osnovu svake od ove četiri kategorije vršiće se pretraga vrednosti atributa `DOID`.

   Svaka kategorija nosi nivo pouzdanosti i tačnosti pronađenih vrednosti atributa `DOID` na osnovu `Xref` vrednosti te kategorije. Samim tim, redosled pretrage korišćenja različitih kategorija je bitan i može doprineti da pronađena vrednost atributa `DOID` bude što preciznija. Redosled kategorija koji je korišćen tokom ovog dela pretraga jeste `E`, `BTNT`, `NTBT` i `Other`. Opis pretrage biće dat nad dve kategorije uz neke dodatne napomene za ostale kategorije koje se odnose na razlike sa opisanim.

   Dakle, pretraga započinje `E` kategorijom `Xref` vrednosti, ova kategorija sastoji se iz `Exact` vrednosti tj. ove `Xref` vrednosti pokazuju na ime bolesti ili `DOID` koji su odgovarajući entitetu u kom je sadržana ta `Xref` vrednost.

   Pomoću metode `GetByOrphaExact` atributske klase `Xrefs` dobija se rečnik u formatu koji je opisan u delu 4.4.2.2. Primenom metode `items` (ugrađena metoda strukture rečnik) na dobijeni rečnik dobija se lista uređenih parova ključ - vrednost. Prvi element uređenog para predstavlja tip `Xref` vrednosti (npr. `Xref.UMLS`), dok drugi element sadrži `Xref` vrednosti tog tipa koje su sačuvane unutar strukture skup (npr. ['C0206307', 'C3542499']).

   Prolaskom kroz listu dobijenu pomoću metode `items` dohvata se tip `Xref` vrednosti, gde se za svaki od tih tipova prolazi kroz skupove njihovih vrednosti. Prolaskom kroz skup ovih vrednosti vrši se pretraga na osnovu njih pomoću metode `GetByXref`, koja je takođe deo atributske klase `Xrefs`. Ukoliko je atribut `DOID` pronađen pretraga se zaustavlja. Primer ovog dela pretrage prikazan je u kodu 4.20.

   Pretraga se nastavlja dalje ukoliko nije pronađena vrednost atributa `DOID`. Nastavak pretrage vrši se na osnovu imena bolesti metodom `GetByDiseaseName` atributske klase `DiseaseName` i to samo u slučaju da je vrednost atributa `DiseaseName` poznata.

   Ukoliko i dalje nije pronađena vrednost atributa `DOID` pretraga se nastavlja

```
exactXrefs = self.annotationContext.xrefs.
    GetByOrphaExact(term.orpha)
for xref, values in exactXrefs.items():
    for value in values:
        doid, doidSource = self.annotationContext.doid.
            GetByXref(xref, value)
        if doid is not None:
            break
    if doid is not None:
        break
```

Listing 4.20: Deo dohvatanja atributa `DOID` pomoću `ORPHACode`-a korišćenjem `E` kategorije `Xref` vrednosti

upotrebom `BTNT` kategorije `Xref` vredosti. Pretraga korišćenjem ove kategorije kompleksnija je u odnosu na pretragu korišćenjem već opisane `Exact` kategorije iz razloga što `Xref` vrednosti ove kategorije ne garantuju da je pronađeni atribut najbolji mogući. Shodno tome, potrebno je proveriti sve `Xref` vrednosti unutar ove kategorije i odabrati najbolji rezultat među ponuđenim.

Pretraga započinje metodom `GetByOrphaBtnt` atributske klase `Xrefs` kojom se dohvata rečnik `BTNT` kategorije u formatu opisanom u delu 4.4.2.2. Prolazak kroz ovaj rečnik i pretraga atributa metodom `GetByXref` odvija se na isti način kao za rečnik kategorije `E`. Kao što je već pomenuto, u delu 4.3.2 `Xref` vrednosti kategorije `BTNT` preslikavaju se na uže pojmove od onih koji su odgovarajući trenutnom entitetu. Iz tog razloga pronađena vrednost atributa `DOID` ne predstavlja konačnu vrednost koja će biti sačuvana unutar anotacijske datoteke. Dohvaćena vrednost atributa `DOID` pomoću metode `GetByXref` koristiće se kao veza prema generalizaciji same te vrednosti zbog osobine `BTNT` `Xref` vrednosti koja je opisana malopre. Takođe, sama generalizacija vrednosti atributa `DOID` je samo kandidat za konačnu vrednost koja će biti sačuvana unutar anotacijske datoteke.

Konačna vrednost će biti izabrana pomoću metrike Žakardov koeficijent sličnosti[13] (eng. *Jaccard similarity coefficient*) koja će se koristiti za poređenje

[13]Žakardov koeficijent sličnosti predstavlja meru sličnosti dva skupa. Sličnost će biti 0 ukoliko dva skupa ne dele nijednu vrednost i 1 ukoliko su dva skupa identična. Merenje se može vršiti nad numeričkim podacima ili nad nekom drugom vrstom podataka.

skupova reči sačinjenih od imena bolesti.

$$jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Dakle, nakon dohvaćene vrednosti atributa `DOID` metodom `GetByXref`, potrebno je korišćenjem metode `GetParentDoidAndDiseaseNamesByDoid` atributske klase `DiseaseName` dohvatiti sve generalizacije dohvaćenog `DOID`-a. Povratna vrednost predstavljena je parom `DOID - Disease Name`, pri čemu vrednost atributa `DOID` predstavlja potencijalnu konačnu vrednost koja će biti sačuvana unutar anotacijske datoteke, a ime bolesti biće iskorišćeno za dobijanje Žakardovog koeficijenta.

Žakardov koeficijent biće izračunat nad skupovima dobijenim od bolesti koja je sadržana u trenutnom redu koji se obrađuje i bolesti koja je dobijena kao generalizacija dohvaćenog `DOID`-a. Atribut `DOID` za koji se bude izračunao najveći Žakardov koeficijent biće sačuvan unutar anotacijske datoteke. Ukoliko trenutni red ne sadrži ime bolesti prva vrednost atributa `DOID` koja bude pronađena biće sačuvana unutar anotacijske datoteke. Primer pretrage korišćenjem `BTNT` kategorije `Xref` vrednosti prikazan je u kodu 4.21.

Ukoliko vrednost atributa `DOID` i dalje nije pronađena naredne dve pretrage koriste kategorije `NTBT` i `Other` na identičan način kao što je opisano za `BTNT` kategoriju. Jedina razlika je u tome što potencijalna vrednost atributa `DOID` koja će biti sačuvana unutar anotacijske datoteke predstavlja vrednost dohvaćenu metodom `GetByXref` atributske klase `DOID` tj. ne vrši se dohvatanje generalizacije ovog atributa zbog svojstva kategorija `NTBT` i `Other`.

Poslednji pokušaj pretrage vrednosti atributa `DOID` vrši se metodom `GetByDiseaseNameUsingSearchEngine` atributske klase `DOID` koja predstavlja pretragu na osnovu pretraživača.

Na samom kraju vrši se pretraga imena bolesti ukoliko se dogodi da neki red ne sadrži ime bolesti, a sadrži vrednost atributa `DOID` pretraga se vrši metodom `GetByDoid` atributske klase `DiseaseName`.

```
    preprocessedDiseaesName = PreprocessingDiseaseName(
                                diseaseName, True)
    if doid is None:
        maxJaccardIndex = -1
        btntXrefs = self.annotationContext.xrefs.GetByOrphaBtnt(
                                                    term.orpha)
        for xref, values in btntXrefs.items():
            for value in values:
                btntDoid, btntDoidSource = self.annotationContext.
                    doid.GetByXref(xref, value)
                parentDoids = self.annotationContext.diseaseName.
                    GetParentDoidAndDiseaseNamesByDoid(btntDoid)
                if preprocessedDiseaesName is not None
                        and btntDoid is not None:
                    for parentDoid in parentDoids:
                        parentDiseaseName = PreprocessingDiseaseName(
                            parentDoid[1], True)
                        currentJaccardIndex = JaccardSimilarity(
                           parentDiseaseName, preprocessedDiseaesName)
                        if currentJaccardIndex > maxJaccardIndex:
                            maxJaccardIndex = currentJaccardIndex
                            doid = parentDoid[0]
                            doidSource = btntDoidSource
                elif preprocessedDiseaesName is None
                        and btntDoid is not None:
                    doid = parentDoids[0][0]
                    if doid is not None:
                        doidSource = btntDoidSource
                        break
            if preprocessedDiseaesName is None and doid is not None:
                break
```

Listing 4.21: Deo dohvatanja atributa `DOID` pomoću `ORPHACode`-a korišćenjem `BTNT` kategorije `Xref` vrednosti

- **Faza IV - Priprema reda za dodavanje u anotacijsku datoteku**

  Nakon što su završene pretrage svih atributa potrebnih za anotacijsku datoteku vrši se kreiranje reda anotacijske datoteke. Redovi anotacijske datoteke opisani su anotacijskim modelom `AnnotationRowOutput` koji nasleđuje osnovi anotacijski model prikazan u kodu 4.1 i sadrži još jedno dodatno polje `DOID Source` koje je opisano u delu 1.3. Kreiranje anotacijskog reda može se podeliti u tri koraka:

1. **Dodatni redovi baze HPO** - prilikom kreiranja anotacijskog reda potrebno je proveriti da li baza `HPO` za jedan red sadrži nekoliko različitih vrednosti za deo veze koji se odnosi na bolest. Ovo se vrši pomoću inicijalizovanog niza `doidAndDiseaseNames` koji sadrži vrednosti atributa `DOID`, `DOID Source` i `Disease Name` kao uređenu trojku. Ukoliko ovaj niz nije prazan potrebno je prolaskom kroz njega dohvatiti sve vrednosti atributa vezanih za bolest i otpakovati ih na način prikazan u kodu 4.22. Bitno je napomenuti da se specijalni slučajevi redova opisani u delu 3.3.3 koji se odnose na nedostajuće vrednosti svih atributa o bolesti pripremaju tokom ovog koraka kreiranja anotacijskog reda.
2. **Specijalni redovi baze Diseases** - u delu 3.3.3 opisan je i slučaj nedostajućih vrednosti svih atributa o genu koji se odnosi na bazu `Diseases`. U kodu 4.23 prikazan je način kreiranja specijalizovanih redova gde se na mesto atributa `Gene Symbol` upisuje identifikator proteina.
3. **Ostali redovi anotacijske datoteke** - kao poslednji korak kreiraju se svi preostali redovi korišćenjem vrednosti atributskih promenljivih. U ovom koraku se anotacijski red kreira od atributskih promenljivih i smešta se u skup `sourceSet`.

Redovi kao anotacijski modeli se dodaju u skup `sourceSet` koji se šalje kao povratna vrednost metode `__ParseSource` zajedno sa imenom baze nad kojom je vršeno parsiranje.

4.5.3 Metod __CreateFullAnnotationSet

Nakon završene pretrage i dohvatanja vrednosti atributa svih baza, anotacijski redovi razvrstani po bazama sačuvani su unutar rečnika `__sourcesAnnotationSetDict`. Primena ovog rečnika opisana je u delu 4.5.1, a objedinjavanje vrednosti samog rečnika vrši se unutar metode `__CreateFullAnnotationSet`. Objedinjavanje se obavlja prolaskom kroz ovaj rečnik dohvatanjem skupova koji sadrže anotacijske redove. Skupovi se dalje nadovezuju na konačni skup koji sadrži anotacijske redove svih baza i naziva se `__annotationSet`. Prikaz metode dat je u kodu 4.24.

```
    if doidAndDiseaseNames:
        for doid, doidSource, diseaseName in doidAndDiseaseNames:
            if doid is None and diseaseName is None and
                    term.omim is not None:
                sourceSet.add(AnnotationRowOutput(symbol, entrezID,
                        uniprotID, ensemblID, doid, sourceName,
                        "<OMIM>" + term.omim, doidSource))
            elif doid is not None or diseaseName is not None:
                sourceSet.add(AnnotationRowOutput(symbol, entrezID,
                    uniprotID, ensemblID, doid, sourceName,
                diseaseName, doidSource))

    elif doid is None and diseaseName is None and
            (term.omim is not None or term.orpha is not None):
        changedDiseaseName = "<OMIM>" + term.omim
            if term.omim is not None else "<ORPHA>" + term.orpha
        sourceSet.add(AnnotationRowOutput(symbol, entrezID, uniprotID,
                    ensemblID, doid, sourceName, changedDiseaseName,
                        doidSource))

    elif doid is not None or diseaseName is not None:
        sourceSet.add(AnnotationRowOutput(symbol, entrezID, uniprotID,
            ensemblID, doid, sourceName, diseaseName, doidSource))
```

Listing 4.22: Dodavanje redova baze `HPO`

```
    elif source is Source.DISEASES and symbol is None and
            term.ensemblProteinID is not None and entrezID is None
            and ensemblID is None and uniprotID is None:
        sourceSet.add(
            AnnotationRowOutput("<PROTEIN_ID>" + term.ensemblProteinID,
                entrezID, uniprotID, ensemblID, doid, sourceName,
                    diseaseName, doidSource))
```

Listing 4.23: Dodavanje specijalizovanih redova baze `Diseases`

4.5.4 Metod CreateAnnotationFile

Poslednji korak kreiranja anotacijske datoteke predstavlja zapis anotacijskih redova na disk. Metod `CreateAnnotationFile` koristi se prilikom kreiranja datoteke

```
    def __CreateFullAnnotationSet(self):
        for source in self.__sources:
            self.__annotationSet |=
                            self.__sourcesAnnotationSetDict[source]
```

Listing 4.24: Metod za objedinjavanje anotacijskih redova `__CreateFullAnnotationSet`

`annotation_file.txt` koja se nalazi unutar direktorijuma `Storage`. Ova datoteka sadrži sve redove anotacijske datoteke koji su sačuvani unutar skupa `__annotationSet`. Kreiranje datoteke se vrši metodom `WriteStructureToFile` koja se koristi prilikom zapisa na disk različitih struktura koje se koriste u aplikaciji `GDA`. Primer metode `CreateAnnotationFile` dat je u kodu 4.25.

```
    def CreateAnnotationFile(self, filePath):
        WriteStructureToFile(filePath,
          sorted(self.__annotationSet, key=lambda term: term.source),
                                                ANNOTATION_FILE_HEADER)
```

Listing 4.25: Metod za kreiranje anotacijske datoteke `CreateAnnotationFile`

4.5.5 Generisanje rezultata tačnosti atributa DOID

Nakon kreirane anotacijske datoteke biće urađena analiza tačnosti atributa `DOID`. Zbog kompleksnosti pronalaženja vrednosti ovog atributa tokom izrade aplikacije kreirana je dodatna funkcija koja daje prikaz tačnosti pronađenih vrednosti atributa `DOID`. `DoidAccuracy` predstavlja funkciju za analizu i prikaz tačnosti vrednosti ovog atributa, prikaz se vrši pomoću tri tabele koje ova funkcija kreira, a opisane su u delu 3.3.2.

Funkcija `DoidAccuracy` prihvata dva argumenta, putanju ka anotacijskoj datoteci i putanju ka datoteci gde će biti sačuvane ove tri tabele. Čitanjem podataka iz anotacijske datoteke kreiraju se spomenute tabele. Nakon prebrojanih redova i izračunatih rezultata, tabele se kreiraju pomoću bilioteke `tabulate` programskog jezika `Python` i čuvaju se unutar datoteke za koju je prosleđena putanja. Takođe,

podatak o ukupnom broju redova će biti sačuvan na kraju iste datoteke. Datoteka `doid_accuracy.txt` biće sačuvana unutar deljenog direktorijuma `Storage` nakon svakog pokrenutog parsiranja i sadržaće pomenute tabele. Dobijeni rezultati predstavljeni su u tabelama 3.1, 3.2 i 3.3. Bitno je napomenuti da se procenti unutar tabela ne sabiraju na 100% zbog zaokruživanja.

Opisane klase i metode u ovom poglavlju implementiraju veliki broj različitih tehničkih elemenata koji nisu opisani ali su vredni spomena. Dodatni elementi predstavljaju merenje vremena izvršavanja celokupnog procesa, prikaz progresa kako ukupnog tako i progresa svake baze pojedinačno. Isto tako, greške prilikom parsiranja biće sačuvane unutar datoteke `error_log.txt` koja se nalazi unutar direktorijuma `Storage`.

4.6 Implementacija interfejsa aplikacije

U ovom delu biće opisana implementacija interfejsa aplikacije koji se koristi za prikaz anotacijske datoteke. Implementacija će biti opisana na nešto višem nivou apstrakcije tj. biće prikazana gruba slika na koji način funkcioniše interfejs aplikacije i šta je korišćeno prilikom izrade interfejsa aplikacije. Dalje, biće predstavljeno na koji način su povezani interfejs aplikacije i serverski deo aplikacije. Implementacija se može razložiti na tri bitna dela: osnova na kojoj je kreiran interfejs aplikacije, implementacija interaktivne tabele i veza između interfejsa aplikacije i serverskog dela aplikacije.

4.6.1 Osnova interfejsa aplikacije

Kao što je već pomenuto interfejs aplikacije kreiran je kao veb aplikacija koja se koristi samo za prikaz anotacijske datoteke. Osnova interfejsa aplikacije kreirana je pomoću razvojnog okvira `django` pisanog u programskom jeziku `Python`. `Django` je besplatni razvojni okvir otvorenog koda koji prati arhitekturu `MVC`[14] [9]. Sadrži objektno relaciono mapiranje[15] (eng. *ORM - Object Relational Mapping*) koje posreduje između `Python` klasa i baze podataka. Kreiranje osnove interfejsa aplika-

[14] `MVC` (eng. *Model-view-controller*) arhitektura je projektni obrazac (eng. *design pattern*) koji se obično koristi za razvoj korisničkih interfejsa. Zasniva se na ideji o ponovnoj upotrebi već postojećeg softverskog koda, olakšava razvoj i održavanje aplikativnog softvera metodom razdvajanja na posebne komponente: model, prikaz podataka (pogled) i kontroler (upravljač).

[15] Mehanizam koji omogućava rad sa klasama i objektima na aplikativnom nivou, uz automatski rad sa bazom podataka.

cije vrši se pomoću ugrađenih `django` komandi[16] koje vrše inicijalizaciju projekta i postavljaju početna podešavanja. Nakon izvršavanja ovih komandi biće generisan kod koji predstavlja `django` projekat. `Django` projekat predstavljen je kolekcijom podešavanja, uključujući konfiguraciju baze podataka, opcije specifične za `django` i podešavanja specifična za aplikaciju. Podrazumevani `django` projekat ne podržava slanje argumenata komandne linije prilikom pokretanja projekta. Shodno tome, za potrebe aplikacije `GDA` kreirana je nova komanda `gda_start` koja se koristi prilikom pokretanja same aplikacije. Ova komanda kao argument komandne linije prima `API_KEY` koji se koristi za povezivanje klijenta i servera integrisanog pretraživača. Prosleđeni ključ se smešta unutar datoteke `api_key.txt` koju klijent pretraživača prilikom inicijalizacije koristi za dohvatanje ključa. Nakon toga vrši se pozivanje podrazumevane komande `runserver` koja startuje aplikaciju. Pri svakom pokretanju biće generisana drugačija vrednost ključa. Prikaz implementacije nove komande dat je u kodu 4.26.

```
class Command(BaseCommand):
    def add_arguments(self, parser):
      parser.add_argument('api_key', type=str)

    def handle(self, *args, **options):
      try:
          with open("api_key.txt", "w") as api_key_file:
              api_key_file.write(options['api_key'])

          call_command("runserver", "0.0.0.0:8000")
      except Exception as e:
          self.stdout.write(self.style.ERROR('ERROR: ' + str(e)))
```

Listing 4.26: Komanda za pokretanje aplikacije `GDA`

[16] Više o samim komandama može se pronaći na linku `https://docs.djangoproject.com/en/4.1/intro/tutorial01/`.

4.6.2 Komunikacija interfejsa aplikacije sa serverskim delom aplikacije

Interfejs aplikacije kao što je opisano u prethodnom delu kreiran je korišćenjem razvojnog okvira `django`, dok je serverski deo aplikacije implementiran korišćenjem samo programskog jezika `Python`. Spajanjem ove dve celine dobija se aplikacija `GDA`. Kao što je već poznato, serverski deo aplikacije zadužen je za kreiranje anotacijske datoteke, dok interfejs aplikacije služi za njen prikaz. Samim tim, ukoliko je kreirana anotacijska datoteka, interfejs aplikacije može da funkcioniše samostalno i ne zahteva nikakvu podršku serverskog dela aplikacije. Već pomenuta arhitektura `MVC` koristi se u `django` projektima, samim tim pogledi (eng. *views*) su deo projekta i predstavljaju metode koje se koriste da obrade određene `HTTP` zahteve. Komunikacija ove dve celine započinje sa `AJAX`[17] (eng. *Asynchronous JavaScript and XML*) zahtevima koji se šalju pomoću `HTTP` protokola kao `GET` ili `POST` metodi. `AJAX` zahtevi predstavljaju jedan od načina komunikacije između osnove interfejsa aplikacije i krajnjeg veb dela koji korisnik vidi i s kojim interaguje. Ove zahteve prihvataju pogledi koji se koriste kao veza između krajnjeg veb dela i serverskog dela aplikacije. Pogledi su i dalje deo interfejsa aplikacije tako da se suštinski nije desila komunikacija sa serverskim delom aplikacije ali je na ovaj način ona započeta jer stvarna komunikacija sa serverskim delom aplikacije biće obavljena unutar pogleda. Klasa zadužena za komunikaciju ove

```
class Parsing:
    @staticmethod
    def parse(progress, initializeSearchEngine=False):
        startTime = time.time()
        parsingContext = ParsingContextThread(progress,
            initializeSearchEngine)
        parsingContext.CreateAnnotationFile(ANNOTATION_PATH)
        DoidAccuracy(ANNOTATION_PATH, DOID_ACCURACY_PATH)
        PrintElapsedTime(startTime, time.time(), "Total elapsed time")
```

Listing 4.27: Klasa `Parsing` zadužena za komunikaciju interfejsa aplikacije i serverskog dela aplikacije

dve celine naziva se `Parsing`. Ova klasa sadrži jedan statički metod `parse` kojim se

[17] `AJAX` pozivi su asinhroni `HTTP` zahtevi i predstavljaju standarni način mrežne komunikacije u `JavaScript`-u. `AJAX` služi da putem `JavaScript`-a dohvati podatke i dinamički ažurira `HTML`, bez ponovnog učitavanja stranice. To je osnovna tehnika za razvoj jednostraničnih aplikacija.

započinje proces kreiranja anotacijske datoteke. Kao argument prima informaciju da li je potrebno izvršiti inicijalizaciju pretraživača. Primer ove klase dat je u kodu 4.27. Celokupan proces komunikacije iz tehničkog ugla biće opisan u narednim koracima:

1. Proces parsiranja započinje se pritiskom na dugme `Parse` što je opisano u delu 3.1. Korišćenjem `JavaScript` koda šalje se `AJAX` zahtev tipa `POST` ka adresi `initialize_parsing/`.

2. Poslati zahtev sa određenom adresom treba nekako da se prosledi dalje ka pogledu koji treba da obradi taj zahtev. Za to je zadužen `URL dispatcher` koji prihvata zahtev, proverava adresu i na osnovu te adrese prosleđuje dalje zahtev ka određenom pogledu.

3. Zahtev spomenut u prvom koraku pomoću `URL dispatcher`-a prosleđuje se ka metodi `initialize_before_parsing` koja predstavlja pogled zadužen za sve zahteve na adresi `initialize_parsing/`. Unutar ovog pogleda vrše se inicijalna podešavanja koja služe za manipulisanje izgledom stranice tokom parsiranja. Nakon izvršene metode pogleda šalje se povratna vrednost `AJAX` zahtevu da je obrađen. Proces parsiranja prati se pomoću `Singleton`[18] klase `FrontendTracker` koja sadrži polja pomoću kojih se ažuriraju elementi veb stranice na kojoj je prikazan proces parsiranja. Na primer, sadrži polje progres koje se ažurira i prikazuje korisniku tokom procesa parsiranja. Prikaz ove klase dat je u kodu 4.28.

4. Nakon obrađenog `AJAX` zahteva iz prvog koraka šalje se novi `AJAX` zahtev takođe tipa `POST` ka adresi `parsing/` i izvršava se `JavaScript` metoda `update` za ažuriranje stanja veb stranice koja će biti opisana u narednom koraku. Pogled zadužen za obradu ovog zahteva izvršava se pomoću metode `parsing_triggered`. Unutar ove metode proverava se da li je korisnik zahtevao inicijalizaciju pretraživača dohvatanjem vrednosti *checkbox*-a `Initialize Search Engine` i započinje se parsiranje pozivanjem statičkog metoda `parse` klase `Parsing`.

5. `JavaScript` metod `update` na svakih `500ms` od momenta kada je poslat zahtev za parsiranje šalje `AJAX` zahteve tipa `POST` ka adresi `update_data/`. Ovi zahtevi obrađuju se u pogledu koji koristi metod `update_data` koji je zadužen za

[18]Unikat (eng. *singleton*) je projektni obrazac kojim se obezbeđuje da klasa ima samo jednu instancu.

```
class FrontendTracker(metaclass=SingletonMetaclass):
    def __init__(self):
        self.parsing = False
        self.progress = 0
        self.parsingStarted = False
        self.spinner = False
        self.showError = False
        self.successParsing = False

    def __call__(self):
        return self
```

Listing 4.28: `Singleton` klasa `FrontendTracker`

ažuriranje polja klase `FrontendTracker`. Kada se parsiranje završi ova metoda prestaje sa slanjem `AJAX` zahteva.

Prethodno navedeni koraci pokazuju način pokretanja kreiranja anotacijske datoteke iz tehničkog ugla. Preostalo je da se pokaže na koji način se dobijena anotacijska datoteka transformiše u interaktivnu tabelu, što će biti opisano u narednom delu.

4.6.3 Transformacija anotacijske datoteke u interaktivnu tabelu

Interaktivna tabela predstavlja tabelu koja sadrži određene podatke kojima je moguće manipulisati na jadnostavan način korišćenjem različitih funkcionalnosti same tabele. Specifično interaktivna tabela aplikacije `GDA` implementirana je korišćenjem dodatka `DataTables` biblioteke `JQuery` programskog jezika `JavaScript`.

Postoji mnogo načina da se dohvate podaci unutar `DataTables` interaktivne tabele. Zbog ogromne količine podataka anotacijske datoteke (200+ hiljada redova) dohvatanje se vrši metodom koja se odvija na serveru (eng. *server-side processing*). Kada je omogućena obrada na strani servera, sve funkcionalnosti interaktivne tabele predaju se serveru gde `SQL engine` (ili neki drugi) može da izvrši ove radnje na velikom skupu podataka. Kao takvo, svako izvlačenje tabele će rezultirati novim `AJAX` zahtevom za dobijanje potrebnih podataka. Samim tim, potrebno je prilagoditi interfejs aplikacije da podrži ovakve zahteve interaktivne tabele.

Kao što je već pomenuto, `django` sadrži podršku za mehanizam `ORM`. Shodno tome, kreiran je model unutar osnove interfejsa aplikacije koji opisuje anotacijsku datoteku. Prikaz modela dat je u kodu 4.29. Tako kreirani model podataka mapira se

```
class AnnotationRowModel(models.Model):
    symbol = models.CharField('symbol', max_length=30,
        default='None')
    entrezID = models.CharField('entrezID', max_length=50,
        default='None')
    uniprotID = models.CharField('uniprotID', max_length=50,
        default='None')
    ensemblID = models.CharField('ensemblID', max_length=50,
        default='None')
    doid = models.CharField('doid', max_length=30, default='None')
    source = models.CharField('source', max_length=30, default='None')
    diseaseName = models.TextField('diseaseName', default='None')
    doidSource = models.CharField('doidSource', max_length=50,
        default='None')
```

Listing 4.29: Klasa `AnnotationRowModel`

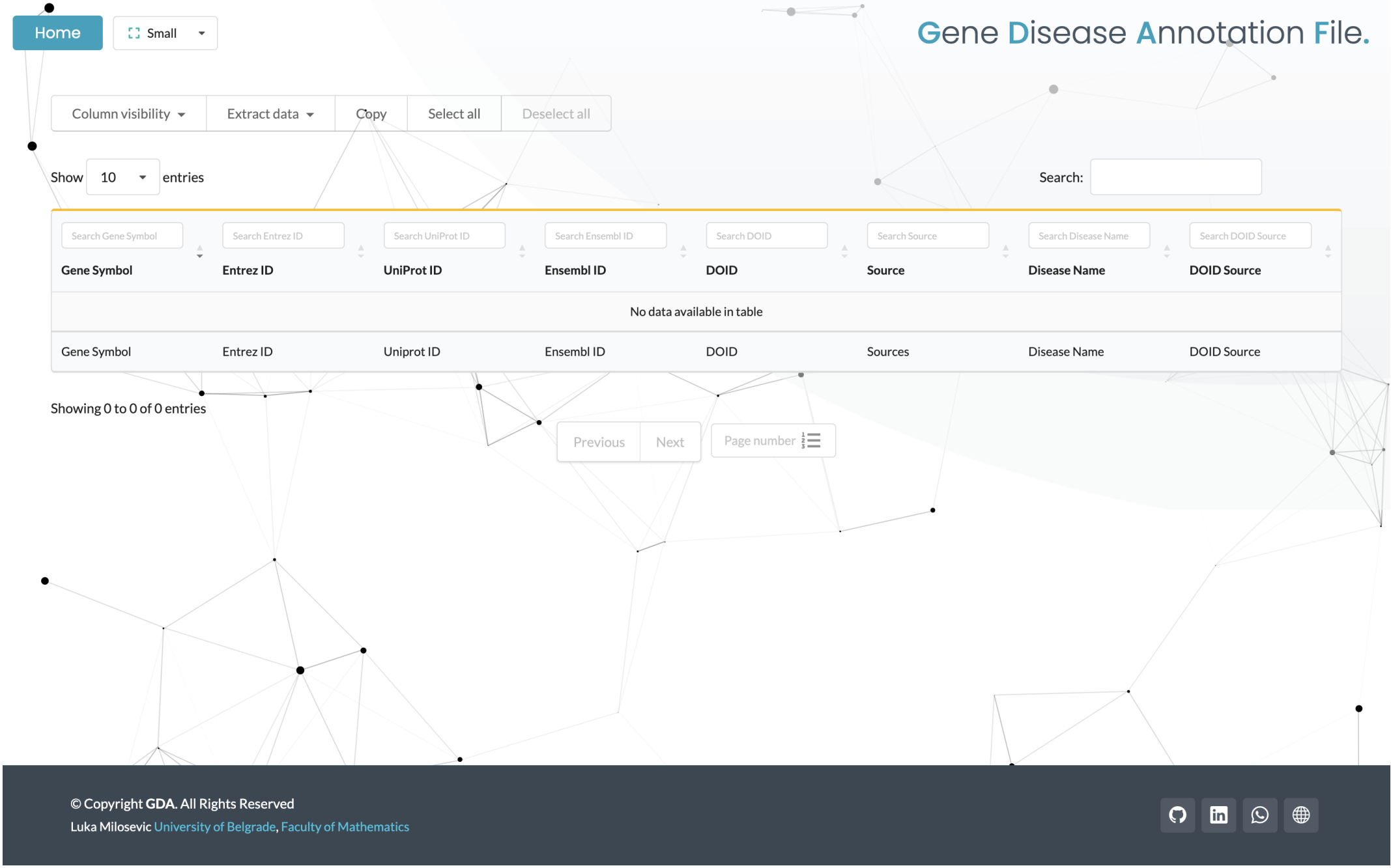
u tabelu baze podataka. Pomoću migracija (eng. *migrations*) vrši se proces kreiranja tabele unutar baze podataka. Migracije predstavljaju način propagiranja promena modela (dodavanje polja, brisanje modela, kreiranje modela, itd.) u šemu baze podataka. Migracije su dizajnirane da budu uglavnom automatske, ali potrebno je i eksplicitno izvršiti neku od migracija.

Prva migracija za kreiranje tabele unutar baze podataka izvršava se prilikom prvog pokretanja aplikacije. Nakon te migracije unutar baze postoji tabela sa poljima koji opisuju anotacijsku datoteku. Tako kreirana tabela ne sadrži podatke, stoga potrebno je popuniti tabelu podacima anotacijske datoteke. Nakon prvog pokretanja aplikacije `GDA` klikom na dugme `See Annotation File` interaktivna tabela neće prikazati nikakve podatke sve dok korisnik ne započne proces kreiranja anotacijske datoteke pritiskom na dugme `Parse`. Prikaz tabele pre prvog pokretanja procesa kreiranja anotacijske datoteke dat je na slici 4.2.

Proces popunjavanja podataka iz anotacijske datoteke vrši se unutar metode `parsing_triggered` pogleda zaduženog za obradu zahteva `parsing/` opisanog u delu 4.6.2. Nakon kreirane anotacijske datoteke, čije je kreiranje započeto meto-

dom `parse` klase `Parsing`, izvršava se dodavanje podataka unutar tabele koja je kreirana prvom migracijom. Dodavanje se vrši narednom migracijom koja tokom izvršavanja prolazi kroz anotacijsku datoteku čije podatke čita i kreira objekte klase `AnnotationRowModel` na osnovu pročitanih podataka. Tako kreirane objekte smešta unutar baze podataka koji će nakon unosa biti prikazani u interaktivnoj tabeli. Sistem za upravljanje bazama podataka koji izvršava delegirane funkcionalnosti od strane interaktivne tabele jeste `SQLite`[19]. Ovaj sistem se zapravo koristi za serversko dohvatanje podataka interaktivne tabele.

[19] `SQLite` je sistem za upravljanje bazama podataka otvorenog koda koji omogućava rad sa relacionim bazama podataka. Male je veličine i lak za podešavanje te je najčešća primena kao lokalno skladište za ugrađene (eng. *embedded*) uređaje i pametne telefone.



Slika 4.2: Tabela nakon prvog pokretanja aplikacije GDA

Glava 5

Pokretanje, izvršavanje i unapređenja

Aplikacija GDA sastoji se od nekoliko celina, pa je i samo pokretanje te aplikacije zahtevan proces. Pokretanje aplikacije zahteva pokretanje serverskog dela pretraživača Typesense i same aplikacije GDA. Ovakav način pokretanja gde se ove dve celine pokreću zasebno zadaje veliki broj problema na jednom operativnom sistemu, a kamoli ako bi postojala želja da aplikacija podržava nekoliko njih. Takođe samo izvršavanje aplikacije predstavlja svojevrstan problem na koji se nailazi ukoliko aplikacija treba da podrži više od jednog operativnog sistema. Shodno tome, u ovom poglavlju biće objašnjeno na koji način aplikacija prevazilazi ove prepreke. Takođe, biće spomenuta neka od unapređenja koja su učinjena tokom razvoja aplikacije GDA.

5.1 Pokretanje i izvršavanje

U ovom delu biće opisano na koji način se pokreće aplikacija GDA. Da bi se ispunili uslovi da pokretanje ne zavisi od operativnog sistema i da kasnije izvršavanje bude podržano na više operativnih sistema, potrebno je koristiti odgovarajuću platformu za takvu podršku. Korišćenjem alata Docker postiže se univerzalan način pokretanja i izvršavanja same aplikacije. Docker je alat dizajniran tako da olakša razvoj, implementaciju i pokretanje aplikacija koristeći kontejnere (eng. *containers*). Softver koji sadrži kontejnere naziva se Docker Engine. Kontejneri su izolovani jedan od drugog i sadrže sopstveni softver, biblioteke i konfiguracione fajlove. Oni mogu međusobno komunicirati putem tačno definisanih kanala. Svim kontejnerima upravlja

kernel operativnog sistema i stoga su jednostavniji od virtualnih mašina. Kontejneri se stvaraju iz slika (eng. *images*) koje određuju njihov precizan sadržaj. Za kreiranje aplikacije GDA koristi se alat Docker Compose koji služi za definisanje i pokretanje Docker aplikacija sa više kontejnera. Ovaj alat koristi YAML datoteke za konfigurisanje usluga aplikacije i izvršava proces kreiranja i pokretanja svih kontejnera sa jednom komandom. Aplikacija GDA koristi još i sistem Docker Volume koji olakšava nezavisnu konzistentnost podataka, omogućavajući podacima da ostanu čak i nakon što se kontejner izbriše ili ponovo kreira.

Nakon opisanih tehničkih elemenata koje aplikacija GDA koristi potrebno je prikazati na koji način je odrađena dokerizacija same aplikacije. Kao što je već pomenuto, aplikacija treba da bude podržana na nekoliko operativnih sistema i to MacOS, Windows i Linux. Alat Docker podržava spomenute operativne sisteme pa će dokerizacija aplikacije omogućiti da i sama aplikacija bude podržana na njima. U narednim koracima biće prikazani osnovni elementi dokerizacije aplikacije GDA.

5.1.1 Kreiranje slike aplikacije GDA korišćenjem Dockerfile-a

Da bi se neka aplikacija izvršavala unutar Docker engine-a potrebno je da bude deo kontejnera. Kontejneri predstavljaju različite instance jedne ili nekoliko slika. Na sliku se može gledati kao na dokerizovanu aplikaciju koja kada se pokrene, pri svakom pokretanju kreira zaseban kontejner koji se izvršava. Kreiranje slika vrši se pomoću Dockerfile-ova koji služe kao nacrt za kreiranje slike. Svaka slika kreira se na način da se uzme neka već postojeća bazna slika nad kojom se vrši nadogradnja u vidu neke aplikacije. Bazna slika koja se koristi za kreiranje slike aplikacije GDA jeste zvanična slika programskog jezika Python. Prilikom kreiranja slike vrši se kopiranje izvornog koda aplikacije GDA, izvršavanje inicijalnih podešavanja i navođenje komande koja će biti pozvana kada se bude pokretala ova aplikacija putem Docker engina-a. U listingu 5.1 dat je prikaz Dockerfile-a koji se koristi za kreiranje slike aplikacije GDA.

```
FROM python:3.9

WORKDIR /GDA

ENV API_KEY ""
```

```
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
ENV PYTHONPATH "${PYTHONPATH}:../"

COPY GDA_backend/Classes ./GDA_backend/Classes
COPY GDA_backend/Common ./GDA_backend/Common
COPY GDA_backend/Data ./GDA_backend/Data
COPY GDA_backend/Other ./GDA_backend/Other
COPY GDA_frontend/Database ./GDA_frontend/Database
COPY GDA_frontend/GDA_datatables ./GDA_frontend/GDA_datatables
COPY GDA_frontend/GDA_frontend ./GDA_frontend/GDA_frontend
COPY GDA_frontend/manage.py ./GDA_frontend
COPY requirements.txt .

RUN ["mkdir", "-p", "./GDA_backend/Storage"]
RUN ["pip3", "install", "-r", "./requirements.txt"]
RUN ["python3", "./GDA_frontend/manage.py", "migrate"]

STOPSIGNAL SIGINT

CMD exec python3 ./GDA_frontend/manage.py gda_start $API_KEY
```

Listing 5.1: `Dockerfile` za kreiranje slike aplikacije `GDA`

Bitno je napomenuti da je prilikom testiranja dokerizovane aplikacije na različitim platformama uočeno da su performanse izvršavanja aplikacije značajno lošije na računarima koji koriste `Intel` čip. Uzrok tog problema je u tome što se kreiranje slike aplikacije vrši na računaru koji koristi `M1`[1] čip, kod kojih se podrazmevano slika kreira za arhitekturu `linux/arm64`. Ovakva arhitektura ne pogoduje računarima koji koriste `Intel` čipove, koji predstavljaju većinu računara na tržištu. Problem je otklonjen kreiranjem slike koja podržava više arhitektura (eng. *multi-arch image*). Arhitekture koje su podržane su `linux/arm64` i `linux/amd64`.

[1] `M1` čip razvijen je od strane kompanije `Apple` i koristi se unutar uređaja razvijenih od strane ove kompanije.

5.1.2 Kreiranje kontejnera za izvršavanje aplikacije GDA

Integracija pretraživača `Typesense` zahteva pokretanje servera koji sadrži kolekcije i izvršava upite nad tim kolekcijama. Shodno tome, server pretraživača je zadužen za obavljanje pretrage i predstavlja glavni deo koji se integriše pored kreiranih klijentskih metoda opisanih u delu 4.4.1.2. `Typesense` pruža besplatnu sliku[2] svog pretraživača koja je dostupna putem `Docker Hub`[3]-a. Isto tako, slika aplikacije `GDA` nalazi se na `Docker Hub`-u i može se pronaći putem sledećeg linka `https://hub.docker.com/r/lukamilosevic11/gda`. Verzija slike `Typesense` pretraživača koja se koristi za potrebe aplikacije `GDA` jeste `0.22.2`. Korišćenjem opisanog alata `Docker Compose` vrši se objedinjavanje ove dve slike tako da funkcionišu kao jedna aplikacija korišćenjem konfiguracione datotke `gda-compose.yml`. Unutar ove datoteke vrši se povezivanje slika aplikacije `GDA` i pretraživača `Typesense` zarad kreiranja kontejnera u kom će ove dve slike funkcionisati kao jedna aplikacija. Primer datoteke `gda-compose.yml` dat je u listingu 5.2.

```
version: "3.9"
services:
  typesense:
    image: typesense/typesense:0.22.2
    container_name: typesense
    logging:
      driver: none
    ports:
      - "8108:8108"
    environment:
      TYPESENSE_API_KEY: ${API_KEY}
      TYPESENSE_DATA_DIR: /data/typesense
    volumes:
      - typesense:/data/typesense
  gda:
    depends_on:
      - typesense
```

[2]Ova slika može se pronaći putem sledećeg linka `https://hub.docker.com/r/typesense/typesense`.

[3]`Docker Hub` predstavlja javno mesto koje pruža `Docker` za pronalaženje i preuzimanje različitih `Docker` slika.

```
    image: lukamilosevic11/gda:latest
    container_name: gda
    environment:
      API_KEY: ${API_KEY}
    volumes:
      - ./Storage:/GDA/GDA_backend/Storage
      - gda:/GDA/GDA_frontend/Database
    ports:
      - "8000:8000"
volumes:
  typesense:
    driver: local
  gda:
    driver: local
```

Listing 5.2: Konfiguraciona datoteka `gda-compose.yml`

Konfiguraciona datoteka `gda-compose.yml` sadrži opis servisa koji se koriste prilikom kreiranja kontejnera. Servisi predstavljaju `Docker` slike koje zahtevaju dodatna podešavanja. Takođe, bitno je napomenuti da `Docker` sadrži mehanizam `Docker Volume` koji pruža mogućnost da se rezultati dobijeni tokom rada aplikacije sačuvaju lokalno na disku, jer svi rezultati zapisani unutar `Docker` kontejnera nestaju nakon prestanka izvršavanja tog kontejnera.

5.1.3 Pokretanje i inicijalna podešavanja dostupna korisniku

Do sada je prikazano nekoliko konfiguracionih datoteka koje se koriste za kreiranje slika ili kontejnera. Sve ove datoteke predstavljaju bitan korak ka konačnom cilju kreiranja aplikacije `GDA`, ali samo nekoliko datoteka je potrebno krajnjem korisniku da bi pokrenuo samu aplikaciju. Pre nego što bude opisano na koji način se pokreće aplikacija `GDA` bitno je spomenuti deljeni direktorijum `Storage`.

5.1.3.1 Deljeni direktorijum Storage

Deljeni direktorijum `Storage` korišćen je od strane već opisanog mehanizma `Docker Volume`. Ovaj direktorijum dostupan je unutar `Docker` kontejnera kao i na lokalnom disku. Sve što ovaj direktorijum sadrži biće prikazano na oba mesta.

Direktorijum `Storage` predstavlja mesto gde će biti sačuvani podaci nakon što se izvršavanje kontejnera završi. Podaci koji se čuvaju unutar ovog direktorijuma su anotacijska datoteka, datoteka zadužena za inicijalizaciju pretraživača i tabele koje prikazuju preciznost atributa `DOID`. Pored toga, moguće je dodati još jednu konfiguracionu datoteku zaduženu za upravljanje bazama koje se parsiraju.

Kao što je poznato, izvorne i pomoćne baze ažuriraju se na nekom vremenskom intervalu. Tokom izrade aplikacije `GDA`, poslednje verzije ovih baza su sačuvane kao deo izvornog koda, pa konfiguraciona datoteka pruža mogućnost korišćenja novijih verzija ovih baza. Konfiguraciona datoteka predstavlja datoteku u `JSON` formatu koja sadrži preslikavanje imena baza na datoteke u kojima su te baze sačuvane. Pomoću ove datoteke moguće je navesti putanju do direktorijuma gde su smeštene nove verzije nekih od baza. Takođe, za svaku od baza je moguće navesti ime datoteke u kojoj je sačuvana ta baza. Ukoliko se neka od baza ne nalazi unutar konfiguracione datoteke biće korišćena baza koja je sačuvana tokom izrade same aplikacije. Konfiguraciona datoteka sačuvana je pod nazivom `data_filenames.json`. Ovaj naziv je bitan jer će unutar aplikacije konfiguraciona datoteka biti pretražena samo po tom nazivu. Primer datoteke dat je u listingu 5.3.

```
{
  "DATA_DIRECTORY": "./Data",
  "CLINVAR": "clinvar.txt",
  "COSMIC": "cosmic.csv",
  "DISEASES": "diseases.tsv",
  "DISGENET": "disgenet.tsv",
  "HPO": "hpo.txt",
  "HUMSAVAR": "humsavar.txt",
  "ORPHANET": "orphanet.xml",
  "RGD_OBO": "rgd.txt",
  "OBO": "obo.txt",
  "UNIPROT": "uniprot.dat",
  "HUGO": "hugo.txt",
  "ORPHANET_XREF": "orphanet_xref.xml",
  "ENSEMBL_ENTREZ": "ensembl_entrez.tsv",
  "ENSEMBL_UNIPROT": "ensembl_uniprot.tsv"
}
```

Listing 5.3: Konfiguraciona datoteka `data_filenames.json`

5.1.3.2 Pokretanje aplikacije GDA

Za pokretanje aplikacije GDA neophodno je imati instalirane alate Docker i Docker Compose što se može učiniti preko zvaničnog Docker sajta koji je dostupan putem linka https://www.docker.com/. Aplikacija GDA dostavlja se kao kompresovani direktorijum pod nazivom GDA.zip. Unutar ovog direktorijuma nalazi se deljeni direktorijum Storage, konfiguraciona datoteka gda-compose.yml, bash skripta GDA.sh i batch skripta GDA.bat. Razlika između ove dve skripte je u tome što se skripta GDA.sh koristi prilikom pokretanja aplikacije na operativnim sistemima Linux i MacOS, dok GDA.bat se koristi za pokretanje na operativnom sistemu Windows. Pokretanje aplikacije će biti opisano posebno za svaku skriptu, dok je lista argumenata koji se koriste identična za obe skripte.

Pokretanje na Windows operativnom sistemu

Prilikom pokretanja aplikacije GDA na operativnom sistemu Windows koristi se skripta GDA.bat. Pokretanje ove skripte vrši se iz aplikacije Command Prompt poznatije kao cmd ili cmd.exe. Za pokretanje skripte potrebno je da korisnik bude pozicioniran unutar direktorijuma GDA gde se i nalazi ova skripta. Komanda koja se koristi za pokretanje skripte koja pokreće samu aplikaciju GDA prati sledeći šablon:

```
GDA.bat <options> [output/reset]
```

Argument <options> predstavlja listu obaveznih argumenata prilikom pokretanja skripte, dok argument [output/reset] predstavlja opcione argumente za dodatne funkcionalnosti prilikom pokretanja aplikacije GDA. Opcioni argument se može primeniti samo na određene obavezne argumente. Lista argumenata će biti prikazana u nastavku i detaljno objašnjena.

Pokretanje na Linux i MacOS operativnim sistemima

Prilikom pokretanja aplikacije GDA na operativnim sistemima Linux i MacOS koristi se skripta GDA.sh. Pokretanje ove skripte vrši se iz terminala. Za pokretanje skripte potrebno je da korisnik bude pozicioniran unutar direktorijuma GDA gde se i nalazi ova skripta. Komanda koja se koristi za pokretanje skripte koja pokreće samu aplikaciju GDA prati sledeći šablon:

```
./GDA.sh <options> [output/reset]
```

Argumenti `<options>` i `[output/reset]` imaju istu ulogu kao i za skriptu koja se koristi prilikom pokretanja aplikacije na `Windows` operativnom sistemu.

Argumenti za pokretanje aplikacije GDA sa primerima komandi

Skripte omogućavaju pokretanje i zaustavljanje aplikacije na nekoliko različitih načina. Za pokretanje se koristi nekoliko obaveznih argumenata bitnih za pokretanje i zaustavljanje aplikacije i dva opciona argumenta zadužena za dodatne funkcionalnosti prilikom pokretanja aplikacije. Svaki argument sadrži kraći i duži oblik (npr. `-help` i `-h`). Kraći oblik će biti prikazan na primeru pokretanja skripte za `Windows`, dok će duži oblik biti prikazan na primeru pokretanja skripte za `Linux` i `MacOS`. Bilo koja kombinacija korišćenja ovih oblika je dozvoljena. Ovi argumenti sa primerima korišćenja biće prikazani i opisani u narednoj listi:

- `-help` ili `-h` - koristi se za prikaz pomoćnog teksta, gde je naveden opis korišćenja skripte.

```
Windows:
    GDA.bat -h
Linux i MacOS:
   ./GDA.sh -help
```

- `-start` ili `-s` - koristi se za pokretanje aplikacije samo ukoliko je kontejner već kreiran.

```
Windows:
    GDA.bat -s
Linux i MacOS:
   ./GDA.sh -start
```

- `-exit` ili `-e` - koristi se za zaustavljanje aplikacije, pritom se ne vrši brisanje kontejnera prilikom zaustavljanja. Ovako zaustavljeni kontejneri mogu se ponovo pokrenuti opisanim argumentom `-s` ili `-start`.

```
Windows:
    GDA.bat -e
Linux i MacOS:
   ./GDA.sh -exit
```

- `-up` ili `-u` - koristi se za pokretanje aplikacije. Ukoliko kontejner ne postoji, on će biti kreiran, a ukoliko postoji, on će biti rekreiran. Ova komanda, pored toga što pokreće aplikaciju, inicijalizuje i `Docker` elemente potrebne za izvršavanje aplikacije. Shodno tome, ukoliko se aplikacija pokreće prvi put, potrebno je koristiti ovu komandu, a ukoliko je kontejner aplikacije iz nekog razloga izbrisan, potrebno je aplikaciju pokrenuti ovom komandom.

```
Windows:
    GDA.bat -u
Linux i MacOS:
   ./GDA.sh -up
```

- `-down` ili `-d` - koristi se prilikom zaustavljanja aplikacije. Kontejner će biti izbrisan prilikom korišćenja ove komande, dok će konekcija sa deljenim direktorijumom `Storage` biti sačuvana.

```
Windows:
    GDA.bat -d
Linux i MacOS:
   ./GDA.sh -down
```

Do sada su bili prikazani obavezni argumenti, a u nastavku će biti prikazana dva opciona argumenta koji se koriste kao dodatak obaveznim argumentima. Opcione argumente je moguće koristiti samo uz prikazane obavezne argumente u narednoj listi:

- `-o` ili `-output` - koristi se za prikaz izvršavanja aplikacije tj. za ispisivanje logova aplikacije unutar terminala u kom je pokrenuta skripta. Ovaj opcioni argument moguće je koristiti samo uz `-up` ili `-u` argumente.

```
Windows:
    GDA.bat -u -o
Linux i MacOS:
   ./GDA.sh -up -output
```

- `-r` ili `-reset` - koristi se za resetovanje aplikacije. Ovim argumentom briše se kontejner i svi prateći elementi potrebni za izvršavanje i kreiranje kontejnera. Ovaj opcioni argument moguće je koristiti samo uz `-down` ili `-d` argumente. Shodno tome, resetovanje je moguće samo nakon pokretanja aplikacije.

```
Windows:
    GDA.bat -d -r
Linux i MacOS:
   ./GDA.sh -down -reset
```

Pokretanje aplikacije nije ograničeno samo na korišćenje skripti. Aplikacija se može pokrenuti kao i svaka druga `Docker` aplikacija korišćenjem `docker-compose` komande, o kojoj se može više naći putem sledećeg linka `https://docs.docker.com/compose/reference/`. Bitno je napomenuti da `docker-compose` komanda prilikom pozivanja konfiguracione datoteke `gda-compose.yml` zahteva inicijalizovanu promenljivu okoline (eng. *environment variable*) `API_KEY`.

Pristup aplikaciji

Nakon što je aplikacija pokrenuta korišćenjem neke od navedenih skripti ili korišćenjem `docker` komande, aplikaciji je moguće pristupiti korišćenjem nekog od pregledača (npr. `Google Chrome`, `Firefox`, `Safari`...). Za pristup aplikaciji nije potrebna Internet konekcija, ali je poželjna zbog određenih elemenata koji se koriste prilikom učitavanja stranica aplikacije (npr. fontovi). Aplikaciji `GDA` se može pristupiti unutar pregledača na adresi `http://127.0.0.1:8000/` ili `http://localhost:8000/`.

5.2 Unapređenja

U ovom delu biće spomenuta određena unapređenja realizovana tokom razvijanja aplikacije `GDA` koja su doprinela boljem funkcionisanju pojedinih delova same aplikacije.

Nalaženje vrednosti atributa `DOID` predstavljao je jedan od zahtevnijih problema prilikom izrade aplikacije `GDA`. Prva verzija algoritma za pronalaženje vrednosti ovog atributa koristila je tehniku grube sile (eng. *brute force*) koja je bila vremenski zahtevna zbog ogromne količine podataka koju je trebalo obraditi. Dešavalo se da pretraga korišćenjem samo jedne baze traje nekoliko desetina minuta. Korišćenje rečnika značajno je ubrazalo kompletan proces pronalaženja vrednosti atributa. Takođe, integrisanje pretraživača doprinelo je brzini same pretrage, gde se pretraga vrši sličnim pristupom kao kod tehnike grube sile upoređivanjem imena bolesti

sa bolestima iz kolekcije ali korišćenjem naprednih tehnika indeksiranja. Korišćenje pretraživača nije doprinelo samo brizini nego i povećanju broja pronađenih vrednosti atributa `DOID`. Pored ova dva unapređenja bitno je spomenuti i korišćenje `Xref` vrednosti čiji je glavni značaj pored brzine i tačnost. Pronađene vrednosti atributa `DOID` korišćenjem `Xref` vrednosti značajno je popravilo procenat potpuno tačno pronađenih vrednosti.

U ranoj fazi implementiranja finalnog parsiranja nije korišćena paralelizacija. Paralelizacija i finalno parsiranje svake baze u zasebnoj niti doprinelo je tome da vreme finalnog parsiranja bude oko dva do tri puta kraće.

Prikazivanje anotacijske datoteke prvobitno je bilo implementirano kao statička `HTML` stranica koja koristi `DataTables` interaktivnu tabelu. Dohvatanje elemenata interaktivne tabele izvršeno je direktnim čitanjem `DOM`[4]-a. Nakon kreiranja anotacijske datoteke izvršena je transformacija podataka u sirovi `HTML`. Pokretanjem statičke veb strane podaci anotacijske datoteke bili bi prikazani unutar interaktivne tabele gde bi se sve funkcionalnosti izvršavale na klijentu. Korišćenjem ovakve implementacije za preko 2000 redova statička veb stranica nije mogla da se učita. Rešenje korišćenjem `django` veb aplikacije i dohvatanjem podataka sa servera značajno je doprinelo performansama interaktivne tabele.

[4]Objektni model dokumenta (eng. *DOM - Document Object Model*) predstavlja hijarhijski prikaz strukture veb stranice, onako kako je vidi pretraživač.

Glava 6

Zaključak

U ovom radu opisana je aplikacija koja omogućava kreiranje anotacijske datoteke i njen prikaz u formi interaktivne tabele pomoću koje je dat pregled veza gen-bolest pri čemu broj pronađenih povezanosti i njihova tačnost najviše zavise od ažurnosti korišćenih baza. Anotacijska datoteka sa trenutnim verzijama baza sadrži podatke za ukupno 294971 veza gen-bolest. Baze koje su korišćene su veoma obimne i mogu sadržati greške u unosima što predstavlja problem jer i najmanja slovna greška može dovesti do lošeg parsiranja koje može da rezultuje lošim poklapanjem bolesti i gena. U skladu sa tim, ni razvijena aplikacija ne može da radi savršeno i da pronađe povezanost za svaku bolest i gen. Osnovna ideja je da aplikacija pomogne pri svakodnevnom radu i olakša korišćenje različitih baza.

Objedinjavanje velikog broja baza predstavlja glavni doprinos ove aplikacije. Objedinjeno je trinaest baza u zajednički format i time je olakšana upotreba informacija koje se nalaze unutar njih. Svaka informacija vezana za gen ili bolest preuzeta iz neke od baza sačuvana je u anotacijskoj datoteci u originalnom formatu iako je vršeno različito pretprocesiranje tako sirovih podataka.

Za poboljšanje aplikacije `GDA` ima dosta prostora, a neka od poboljšanja su navedena u narednoj listi:

- korišćenje tehnika mašinskog učenja, s obzirom na njihovu ekspanziju u poslednje vreme.
- korišćenje nekog drugog programskog jezika može u velikoj meri doprineti optimizaciji algoritama predstavljenih u radu.
- mogućnost da se proces kreiranja anotacijske datoteke obogati dodatnim bazama koje nisu razmatrane.

Kao što je već pomenuto, aplikacija će biti besplatna i javno dostupna. Izvorni kod aplikacije može se pronaći putem sledećeg linka `https://github.com/lukamilosevic11/GDA`.

Bibliografija

[1] Blaise TF Alako, Antoine Veldhoven, Sjozef van Baal, Rob Jelier, Stefan Verhoeven, Ton Rullmann, Jan Polman, and Guido Jenster. Copub mapper: mining medline based on search term co-publication. *BMC Bioinformatics*, 6(1):51, Mar 2005.

[2] Michael Ashburner, CJ Mungall, and SE Lewis. Ontologies for biologists: a community model for the annotation of genomic data. In *Cold Spring Harbor symposia on quantitative biology*, volume 68, pages 227–236, 2003.

[3] Ewan Birney, T Daniel Andrews, Paul Bevan, Mario Caccamo, Yuan Chen, Laura Clarke, Guy Coates, James Cuff, Val Curwen, Tim Cutts, Thomas Down, Eduardo Eyras, Xose M Fernandez-Suarez, Paul Gane, Brian Gibbins, James Gilbert, Martin Hammond, Hans-Rudolf Hotz, Vivek Iyer, Kerstin Jekosch, Andreas Kahari, Arek Kasprzyk, Damian Keefe, Stephen Keenan, Heikki Lehvaslaiho, Graham McVicker, Craig Melsopp, Patrick Meidl, Emmanuel Mongin, Roger Pettett, Simon Potter, Glenn Proctor, Mark Rae, Steve Searle, Guy Slater, Damian Smedley, James Smith, Will Spooner, Arne Stabenau, James Stalker, Roy Storey, Abel Ureta-Vidal, K Cara Woodwark, Graham Cameron, Richard Durbin, Anthony Cox, Tim Hubbard, and Michele Clamp. An overview of ensembl. *Genome Res.*, 14(5):925–928, May 2004.

[4] Jared M Bischof, Annie P Chiang, Todd E Scheetz, Edwin M Stone, Thomas L Casavant, Val C Sheffield, and Terry A Braun. Genome-wide identification of pseudogenes capable of disease-causing gene conversion. *Hum. Mutat.*, 27(6):545–552, June 2006.

[5] HUGO Gene Nomenclature Committee. About the HGNC. on-line at: `https://www.genenames.org/about/` (septembar 2022.).

[6] COSMIC. What is COSMIC? on-line at: `https://cancer.sanger.ac.uk/cosmic/about` (septembar 2022.).

[7] Wilco W. M. Fleuren, Stefan Verhoeven, Raoul Frijters, Bart Heupers, Jan Polman, René van Schaik, Jacob de Vlieg, and Wynand Alkema. CoPub update: CoPub 5.0 a text mining system to answer biological questions. *Nucleic Acids Research*, 39(suppl_2):W450–W454, 05 2011.

[8] Marti Hearst. What Is Text Mining?, 2003. on-line at: `https://people.ischool.berkeley.edu/~hearst/text-mining.html` (septembar 2022.).

[9] A. Holovaty and J. Kaplan-Moss. *The Definitive Guide to Django: Web Development Done Right*. IT Pro. Apress, 2009.

[10] Andreas Hotho, A. Nürnberger, and Gerhard Paass. A brief survey of text mining. *LDV Forum*, 20:19–62, 2005.

[11] European Bioinformatics Institute. Investigating a gene, 1992. on-line at: `https://www.ebi.ac.uk/training/online/courses/ensembl-browsing-genomes/navigating-ensembl/investigating-a-gene/` (septembar 2022.).

[12] Rebecca Jackson, Nicolas Matentzoglu, James A Overton, Randi Vita, James P Balhoff, Pier Luigi Buttigieg, Seth Carbon, Melanie Courtot, Alexander D Diehl, Damion M Dooley, William D Duncan, Nomi L Harris, Melissa A Haendel, Suzanna E Lewis, Darren A Natale, David Osumi-Sutherland, Alan Ruttenberg, Lynn M Schriml, Barry Smith, Christian J Stoeckert Jr., Nicole A Vasilevsky, Ramona L Walls, Jie Zheng, Christopher J Mungall, and Bjoern Peters. OBO Foundry in 2021: operationalizing open data principles to evaluate ontologies. *Database*, 2021, 10 2021. baab069.

[13] Melissa J Landrum, Shanmuga Chitipiralla, Garth R Brown, Chao Chen, Baoshan Gu, Jennifer Hart, Douglas Hoffman, Wonhee Jang, Kuljeet Kaur, Chunlei Liu, Vitaly Lyoshin, Zenith Maddipatla, Rama Maiti, Joseph Mitchell, Nuala O'Leary, George R Riley, Wenyao Shi, George Zhou, Valerie Schneider, Donna Maglott, J Bradley Holmes, and Brandi L Kattman. ClinVar: improvements to accessing data. *Nucleic Acids Res.*, 48(D1):D835–D844, January 2020.

[14] Orphanet. FREE ACCESS PRODUCTS DESCRIPTION, 2021. on-line at: `http://www.orphadata.org/cgi-bin/img/PDF/OrphadataFreeAccessProductsDescription.pdf` (septembar 2022.).

[15] Janet Piñero, Juan Manuel Ramírez-Anguita, Josep Saüch-Pitarch, Francesco Ronzano, Emilio Centeno, Ferran Sanz, and Laura I Furlong. The DisGeNET knowledge platform for disease genomics: 2019 update. *Nucleic Acids Research*, 48(D1):D845–D855, 11 2019.

[16] Sune Pletscher-Frankild, Albert Pallejà, Kalliopi Tsafou, Janos X. Binder, and Lars Juhl Jensen. Diseases: Text mining and data integration of disease–gene associations. *Methods*, 74:83–89, 2015.

[17] Sue Richards, Nazneen Aziz, Sherri Bale, David Bick, Soma Das, Julie Gastier-Foster, Wayne W Grody, Madhuri Hegde, Elaine Lyon, Elaine Spector, Karl Voelkerding, Heidi L Rehm, and ACMG Laboratory Quality Assurance Committee. Standards and guidelines for the interpretation of sequence variants: a joint consensus recommendation of the american college of medical genetics and genomics and the association for molecular pathology. *Genet. Med.*, 17(5):405–424, May 2015.

[18] ROSALIND. Gene Symbol. on-line at: `https://rosalind.info/glossary/gene-symbol/` (septembar 2022.).

[19] Lynn Marie Schriml, Cesar Arze, Suvarna Nadendla, Yu-Wei Wayne Chang, Mark Mazaitis, Victor Felix, Gang Feng, and Warren Alden Kibbe. Disease ontology: a backbone for disease semantic integration. *Nucleic Acids Res.*, 40(Database issue):D940–6, January 2012.

[20] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J Mungall, et al. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255, 2007.

[21] Jennifer R Smith, G Thomas Hayman, Shur-Jen Wang, Stanley J F Laulederkind, Matthew J Hoffman, Mary L Kaldunski, Monika Tutaj, Jyothi Thota, Harika S Nalabolu, Santoshi L R Ellanki, Marek A Tutaj, Jeffrey L De Pons, Anne E Kwitek, Melinda R Dwinell, and Mary E Shimoyama. The year of the

rat: The rat genome database at 20: a multi-species knowledgebase and analysis platform. *Nucleic Acids Res.*, 48(D1):D731–D742, January 2020.

[22] Yoshimasa Tsuruoka, Makoto Miwa, Kaisei Hamamoto, Jun'ichi Tsujii, and Sophia Ananiadou. Discovering and visualizing indirect associations between biomedical concepts. *Bioinformatics*, 27(13):i111–i119, 06 2011.

[23] Yoshimasa Tsuruoka, Jun'ichi Tsujii, and Sophia Ananiadou. FACTA: a text search engine for finding associated biomedical concepts. *Bioinformatics*, 24(21):2559–2560, 09 2008.

[24] Typesense. About Typesense. on-line at: `https://typesense.org/about/` (septembar 2022.).

[25] UniProt. About UniProt. on-line at: `https://www.uniprot.org/help/about` (septembar 2022.).

[26] Elio F. Vanin. Processed pseudogenes: Characteristics and evolution. *Annual Review of Genetics*, 19(1):253–272, 1985. PMID: 3909943.

[27] Jaie Woodard, Chengxin Zhang, and Yang Zhang. Address: A database of disease-associated human variants incorporating protein structure and folding stabilities. *Journal of Molecular Biology*, 433(11):166840, 2021. Computation Resources for Molecular Biology.

Biografija autora

Luka Milošević rođen je 11. marta 1996. godine u Beogradu. Završio je Osnovnu školu „Braća Baruh" 2011. godine u Beogradu, nakon toga upisao je srednju Elektrotehničku školu „Nikola Tesla" u Beogradu, smer elektrotehničar računara. Odmah po završetku srednje škole, upisao je osnovne akademske studije na Matematičkom fakultetu u Beogradu, modul Informatika. Diplomirao je u roku 2019. godine sa prosekom 8.90 i stekao zvanje Diplomirani informatičar. Obrazovanje je nastavio odmah na istom fakultetu, gde u oktobru 2019. godine upisuje master studije, pod istim programom studija. Zaključno sa septembrom 2020. godine, položio je sve ispite na master studijama sa prosekom 9.83, a u septembru 2022. završio je izradu svog master rada pod naslovom *„Razvoj aplikacije za integraciju podataka iz različitih izvora o povezanosti gena i bolesti"*. Od novembra 2020. godine, zvanično je započeo svoju profesionalnu karijeru kao softverski inženjer.