Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - 141501

QUESTION ANSWERING SYSTEM DENGAN PENDEKATAN TEMPLATE BASED BERBASIS LINKED DATA PADA DATA KANKER PROSTAT PADA PLATFORM TELEGRAM

QUESTION ANSWERING SYSTEM USING TEMPLATE BASED APPROACH BASED ON LINKED DATA OF PROSTATE CANCER DATA ON TELEGRAM PLATFORM

NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP 5213100078

Dosen Pembimbing
Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

Halaman ini sengaja dikosongkan

TUGAS AKHIR - 141501

QUESTION ANSWERING SYSTEM DENGAN PENDEKATAN TEMPLATE BASED BERBASIS LINKED DATA PADA DATA KANKER PROSTAT PADA PLATFORM TELEGRAM

NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP 5213100078

Dosen Pembimbing
Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESIS - 141501

QUESTION ANSWERING SYSTEM USING TEMPLATE BASED APPROACH BASED ON LINKED DATA OF PROSTATE CANCER DATA ON TELEGRAM PLATFORM

NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP 5213100078

Supervisor
Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D

DEPARTMENT OF INFORMATION SYSTEM
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2017

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

QUESTION ANSWERING SYSTEM DENGAN PENDEKATAN TEMPLATE BASED BERBASIS LINKED DATA PADA DATA KANKER PROSTAT PADA PLATFORM TELEGRAM

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Akuisisi Data dan Diseminasi Informasi
Program Studi S1 Departemen Sistem Informasi
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :

NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP: 5213100078

Surabaya, September 2017

KEPALA
DEPARTEMEN SISTEM INFORMASI

Dr. Ir. Ahs Tjahyanto, M.Kom.
NIP. 19650310 199102 1 001

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

QUESTION ANSWERING SYSTEM DENGAN PENDEKATAN TEMPLATE BASED BERBASIS LINKED DATA PADA DATA KANKER PROSTAT PADA PLATFORM TELEGRAM

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Akuisisi Data dan Diseminasi Informasi
Program Studi S1 Departemen Sistem Informasi
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :

NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP: 5213100078

Disetujui Tim Penguji: Tanggal Ujian: 10 Juli 2017
Periode Wisuda: September 2017

Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D (Pembimbing 1)

Renny Pradina K, S.T, M.T, SCJP **(Penguji 1)**

Radityo Prasetianto W., S.Kom, M.Kom **(Penguji 2)**

Halaman ini sengaja dikosongkan

QUESTION ANSWERING SYSTEM DENGAN PENDEKATAN TEMPLATE BASED BERBASIS LINKED DATA PADA DATA KANKER PROSTAT PADA PLATFORM TELEGRAM

Nama	: NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP	: 5213100078
Jurusan	: Sistem Informasi FTIf
Pembimbing I	: Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D

Abstrak

Informasi kesehatan adalah satu dari beberapa informasi yang sering dicari di internet. Untuk mendapatkan informasi yang baik dan terpercaya sangat penting untuk memperhatikan kelengkapan dari informasi dan integrasi data. Semantic Web merupakan salah satu cara untuk menghubungkan data yang dibuat dalam suatu file RDF. Semantic Web yang telah dibangun untuk memberikan informasi mengenai kesehatan adalah BeinWell. BeinWell mengintegrasikan 5 dataset RDF agar pengguna dapat mengakses informasi secara luas. Di sisi lain, pengembangan dari Semantic Web BeinWell, sangat bergantung pada penggunaan SPARQL Query secara penuh agar pengguna mendapatkan informasi yang diinginkan dan penggunaan keyword untuk mencari informasinya. Oleh karena itu dibuatkan sebuah sistem yang mampu melakukan konversi terhadap bahasa yang digunakan manusia sehari-hari atau Natural Language menjadi SPARQL Query. Dengan sebuah Question Answering System ini, pengguna tidak perlu mengetikan bahasa query yang panjang, melainkan cukup hanya menggunakan bahasa inggris yang baik dan benar. Question Answering System ini dibangun dengan memanfaatkan pengembangan teknologi NLP, seperti POS Tagging, Wordnet dan algoritma Levenshtein Distance. Penggunaan NLP pada Question Answering System ini dimak-

sudkan untuk meningkatkan pemahaman semantik sistem terhadap sebuah pertanyaan, yang akan dimanfaatkan pada sebuah templat query yang dibangun. Berdasarkan hasil penelitian ini dihasilkan berupa Question Answering System natural language question ke SPARQL Query yang telah diimplementasikan dalam sebuah Chat bot pada platform Telegram. Sistem query yang dibangun dengan menggunakan kombinasi teknologi NLP ini mampu bekerja dengan ketepatan informasi yang diambil sebesar 88,88 persen dengan kecepatan proses mencapai 4,8 detik untuk merespon pertanyaan

Kata kunci: Linked Data, Kanker Prostate, Semantic Search, Konversi, NLP, SPARQL.

QUESTION ANSWERING SYSTEM USING TEMPLATE BASED APPROACH BASED ON LINKED DATA OF PROSTATE CANCER DATA ON TELEGRAM PLATFORM

Name	: NIKOLAUS HERJUNO SAPTO DWI ATMOJO
NRP	: 5213100078
Major	: Information System FTIf
Supervisor I	: Nur Aini Rakhmawati, S.Kom., M.Sc.Eng, Ph.D

Abstract

Health information is one of the most frequently searched information on the internet. To get good and reliable information is very important to pay attention to the completeness of information and data integration. Semantic Web is one way to connect data created in an RDF file. A variety of semantic web to deliver health information that has been built, such as the BeinWell website. BeinWell integrates 5 RDF datasets for users to access information widely. On the other hand, the development of BeinWell's Semantic Web relies heavily on full SPARQL Query usage so that users get the desired information and keyword usage to find the information. Therefore created a system that is able to convert the language used everyday people or Natural Language into SPARQL Query. With a conversion system, users do not need to type a long query language, but simply use only good and correct English. This conversion system was built by utilizing the development of NLP technology, such as POS Tagging, Wordnet and Levenshtein Distance algorithms. The use of NLP in the conversion system is intended to increase the system's semantic understanding of a question, which will be exploited in a built-in query template. Based on the results of this study resulted in a natural language question conversion system to SPARQL Query that has been implemented in a Chat bot on the Telegram platform. The query system built using a combination of NLP tech-

nology is able to work with the accuracy of the information taken for 90 percent

Keywords: Linked Data, Kanker Prostate, Semantic Search, Konversi, NLP, SPARQL.

KATA PENGANTAR

Segala puji dan syukur pada Tuhan Yang Maha Esa yang telah melimpahkan rahmat dan anugerah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "Question Answering System dengan Pendekatan Template Based Berbasis Linked Data Pada Data Kanker Prostat Pada Platform Telegram" dengan tepat waktu.

Harapan dari penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini, serta dapat memberikan kontribusi nyata bagi kampus Sistem Informasi, ITS, dan bangsa Indonesia.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin menyampaikan terimakasih kepada:

1. Ibu Nur Aini Rakhmawati, S.Kom., M.Sc., Eng. selaku dosen pembimbing penulis yang telah memberikan ide, bimbingan, saran, kritik, ilmu, dan pengalamannya yang sangat bermanfaat sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Seluruh dosen Departemen Sistem Informasi ITS yang telah memberikan ilmu pengetahuan dan pengalaman yang sangat berharga dan bermanfaat bagi penulis.
3. Seluruh keluarga besar saya khususnya Bapak, Ibu serta Kakak dan Adik saya yang telah memberikan semangat serta kepercayaan kepada saya dalam mengerjakan tugas akhir
4. Teman-teman dan Sahabat terdekat saya angkatan 2013 (13ELTRANIS) yang senantiasa menemani daam prosses pengerjaan serta diskusi yang diberikan.
5. Sahabat terdekat saya, SAHABAT SAMBAT, yang menjadi tempat curahan hati dan cerita selama pengerjaan Tugas

Akhir

6. Rekan-rekan anggota Laboratorium Akuisisi Data dan Diseminasi Informasi atas segala pencerahan, inspirasi dan bantuan yang diberikan
7. Rekan-rekan dari Laboratorium lain yang memberikan hiburan ketika berkunjung ke Laboratorium ADDI untuk menumpang mengerjakan Tugas Akhir.
8. Rekan-rekan organisasi Himpunan Mahasiswa Sistem Informasi HMSI ITS yang telah memberikan pengalaman, pelajaran berharga dan bermanfaat selama disana.
9. Serta seluruh pihak-pihak lain yang tidak dapat disebutkan satu per satu yang telah banyak membantu penulis selama perkuliahan hingga dapat menyelesaikan tugas akhir ini.

Tugas Akhir ini merupakan persembahan bagi penulis untuk kedua orang tua dan keluarga besar yang selalu memberikan motivasi terbaik bagi penulis untuk dapat menuntut ilmu setinggi-tingginya dan dapat meraih kesuksesan.

Tugas Akhir ini juga masih jauh dari kata sempurna, sehingga penulis mengharapkan saran dan kritik yang membangun dari pembaca untuk perbaikan ke depan. Semoga Tugas Akhir ini dapat bermanfaat bagi perkembangan ilmu pengetahuan dan semua pihak.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE

BAB 1

PENDAHULUAN

Pada bab pendahuluan akan diuraikan proses identifikasi masalah penelitian yang meliputi latar belakang masalah, perumusan masalah, batasan masalah, tujuan tugas akhir, manfaat kegiatan tugas akhir dan relevansi terhadap pengerjaan tugas akhir. Berdasarkan uraian pada bab ini, harapannya gambaran umum permasalahan dan pemecahan masalah pada tugas akhir dapat dipahami.

1.1 Latar Belakang

Era Web 3.0 yang diperkenalkan oleh World Wide Web merubah paradigma penggunaan website oleh penggunanya. Sering disebut bahwa era tersebut merupakan lahirnya Era Web of Data dan erat kaitannya dengan terminologi Linked Data. Linked Data telah membuat sebuah web dapat saling meng- hubungkan informasi ataupun data yang berasal dari beragam source, agar dapat dimanfaatkan oleh pengguna lain untuk memperkaya informasi web mereka [10]. Dengan semakin terhubungnya data dan informasi dari satu source dengan source lain, maka akan melahirkan potensi bahwa aksesibilitas pengguna terhadap sebuah informasi akan semakin luas [10].

Di samping itu dengan terjadinya ekstensibilitas data, sebuah metode untuk mengakses informasi secara cepat dan tepat perlu dipertimbangkan. Menurut (Ferret et al., 2001) penggunaan keyword dalam melakukan pencarian pada search engine yang mempertimbangkan popularity measures, keyword matching, frequencies of accessing documents, serta aspek lainnya ternyata tidak sepenuh-

nya menyelesaikan ataupun memenuhi kebutuhan pengguna dalam mendapatkan informasi [11] . Pengguna diharuskan memeriksa satu per satu dokumen untuk mendapatkan informasi yang diharapkan, di mana hal ini membuat proses pengambilan informasi atau information retrieval menjadi sangat memakan waktu [11] . Di mana pada kasus ini, information retrieval seharusnya memberikan lebih sedikit relevant document atau bahkan di tingkat frasa kalimat.

Searching merupakan titik mula dari sebuah informasi untuk dikuak. Terdapat sebuah survey yang dilakukan oleh Kelton Research terhadap perilaku pengguna terhadap website. Kesalahan yang dialami oleh pengguna ketika mengakses informasi dalam navigasi website, sekitar 50 persen pengguna akan cenderung menuju search ketika tidak dapat menemukan informasi yang diinginkan [4]. Sedangkan 71 persen pengguna e-commerce akan cenderung menggunakan fitur search untuk menemukan produk yang diinginkan [4]. Dari sisi penggunaannya sendiri, bahwa 65.4 persen orang merasa terbuang waktunya ketika melakukan search pada website [4]. Dari sini dapat disimpulkan bahwa penggunaan term search berbasis keyword (keyword-based) dalam melakukan search. Hal ini diakibatkan karena keyword searching mengalami kesulitan ketika membedakan kata-kata yang memiliki ejaan yang sama namun memiliki arti yang berbeda. Search sendiri memiliki 2 tipe search, yaitu basic search dan advance search, namun nyatanya pengguna jarang menggunakan advance search karena pada dasarnya advance search menggunakan word pairing [4] [8]. Hal ini lah yang mengakibatkan hasil search yang buruk dan hasil yang tidak akurat.

Keyword-based pada penggunaan search mulai digantikan pada pengembangan Natural Language sebagai alternatif sebuah search. Pengembangan ini mulai ditemukan sebagai Semantic Search, di mana pengembangan ini dengan membentuk suatu interface terhadap Natural Language [13]. Semantic Search bertujuan untuk meningkatkan akurasi search dengan lebih memahami kontekstual

serta arti dari terminologi search pengguna dibandingkan dengan Keyword Search [8]. Hal ini karena Keyword Search cenderung memberikan hasil kembalian dokumen yang banyak namun dengan hasil yang tidak relevan [4]. Sedangkan Semantic Search mampu memberi kembalian dalam bentuk frasa terhadap Natural Language pengguna. Natural Language Search atau Semantic Search berfokus pada arti mengenai bagaimana seorang manusia bertanya senatural mungkin [13] [4]. Hal ini karena Natural Language merupakan Concept-based.

Sama seperti website lain yang mengimplementasikan search untuk memudahkan penggunanya dalam menemukan informasi, BeinWell sebagai semantic web yang memuat integrasi antara 5 data sources mengenai informasi medikal serta kanker prostat pada manusia pun memiliki data serta informasi yang sangat luas untuk diakses. Pengimplementasian Natural Language Question terhadap search ditujukan untuk memudahkan pengguna dalam menemukan informasi yang relevan dalam jumlah yang tidak terlalu banyak, sehingga pengguna tidak perlu menghabiskan banyka waktu dalam menemukan informasi yang diinginkannya.

1.2 Rumusan Masalah

Berdasarkan uraian latar belakang, maka rumusan permasalahan yang menjadi fokus dan akan diselesaikan dalam Tugas Akhir ini antara lain :

1. Bagaimana membangun Question Answering System untuk menjawab pertanyaan seputar kanker prostat pada data RDF menggunakan metode Template Based?
2. Bagaimana membangun Question Answering System pada platform Telegram

3. Bagaimana kinerja Question Answering System dalam memberikan jawaban secara benar?

1.3 Batasan Masalah

Dari permasalahan yang disebutkan di atas, batasan masalah dalam tugas akhir ini adalah :

1. Tugas akhir ini hanya sebatas konversi natural language menjadi SPARQL query.
2. Metode yang digunakan dalam mengkonversi natural language menjadi SPARQL query adalah *Template-Based*
3. Tipe pertanyaan yang didukung adalah ”What”, ”Where”, ”When”, dan ”Who”.
4. Natural Language Question menggunakan bahasa Inggris.

1.4 Tujuan

Berdasarkan pemaparan dan penuturan rumusan masalah serta batasan masalah sebelumnya, maka tujuan dari pembuatan tugas akhir ini adalah membangun Sistem Konversi Natural Language Question ke SPARQL Query terhadap RDF Data Kanker Prostat

1.5 Manfaat

Manfaat yang akan diperoleh dengan tugas akhir ini antara lain:

1. Dapat mengembangkan serta penyebarluasan pengetahuan mengenai Natural Language Processing.

2. Dapat mengembangkan serta penyebarluasan pengetahuan mengenai Linked Data dan RDF.
3. Memfasilitasi pengguna yang ingin mendapatkan informasi mengenai kanker prostat secara cepat dan spesifik.

1.6 Relevansi

Tugas akhir ini berkaitan dengan mata kuliah Konstruksi Pengembangan Perangkat Lunak, Pemrograman Integratif dan Teknologi Open Source dan Terbarukan.

Halaman ini sengaja dikosongkan

BAB 2

TINJAUAN PUSTAKA

Pada bab ini dijelaskan mengenai teori-teori terkait yang bersumber dari buku, jurnal, ataupun artikel yang berfungsi sebagai dasar dalam melakuk an pengerjaan tugas akhir agar dapat memahami konsep atau teori penyelesaian permasalahan yang ada.

2.1 Penelitian sebelumnya

Beberapa penelitian sebelumnya terkait baik Natural Language Question maupun Question Answer System terhadap Semantic Web akan dibahas sebagai berikut:

1. Template-based Question Answering over RDF Data [15] . Tujuan dari penelitian ini adalah untuk meningkatkan hasil pengambilan informasi berupa asnwer pada struktur semantik dari *natural language question*. Hal ini dilakukan dengan menghadirkan pendekatan yang mengubah atau parsing sebuah question untuk menghasilkan SPARQL Template yang secara langsung ditujukan kepada internal struktur pertanyaan. Di mana SPARQL template kemudian akan diisikan sebuah instance menggunakan *statistical entity detection* dan predicate atau property detection. RDF yang digunakan berasal dari DBpedia yang merupakan RDF data dari Wikipedia. Penggunaan Stanford Named Entity digunakan untuk mengimplementasikan *Entity Detection* dan *Predicate Detection*, di mana kemudian akan dilakukan *query ranking* dan pemilihan query untuk mendapatkan query yang menghasilkan result terbaik. Namun ditemukan beberapa kekurangan di

mana baik Stanford POS Tagger dan Apache OpenNLP POS Tagger masih memberikan hasil yang ambigu terhadap beberapa struktur kalimat, sehingga terdapat beberapa pertanyaan yang tidak di-*parse*. Melakukan training terhadap model POS Tagger dapat dilakukan untuk dapat mengatasi banyak model pertanyaan.

2. Question Answering System, Approaches and Techniques: A Review [12] . Tujuan dari penelitian ini adalah memberikan review atau pembahasan mengenai pendekatan-pendekatan yang umum digunakan dalam mengembangkan *Question Answering System* beserta dengan tantangan dalam pengembangannya. Penelitian ini menjelaskan framework QAS dibangun menggunakan *Natural Language Processing* dan *Information Retrieval Techniques* . *Framework* tersebut dibagi menjadi 4 modul, *Question processing Module*, *Document Processing Module*, *Paragraph extraction module* dan *Answer Extraction module*. Beberapa pendekatan yang digunakan adalah *Linguistic-based Approach*, *Statistical-based Approach* dan *Pattern matching Approach*. *Linguistic-based Approach* memanfaatkan pemahaman terhadap natural language text, bahasa dan teknik-teknik seperti tokenisasi, POS Tagging dan parsing. Teknik ini digunakan pada question pengguna untuk memformulasikan query yang tepat dari *structured database*. *Statistical-based Approach* memanfaatkan penggunaan teknik-teknik statistical learning seperti Bayesian Classifier, *Support Vector Machine Classifier* dan *Maximum Entropy Model* . Sedangkan yang terakhir adalah *Pattern matching Approach* memanfaatkan pemahaman terhadap pattern atau pola terhadap question pengguna, beberapa Question Answer yang menggunakan *Pattern matching* menggunakan *surface text pattern* dan ada juga yang mengandalkan template-template pertanyaan yang sudah didefinisikan sebelumnya untuk merespon question pengguna. Bebe-

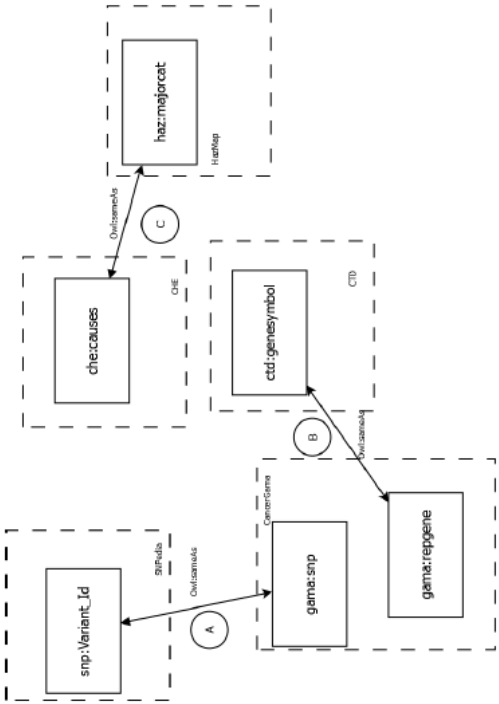
rapa isu serta tantangan pada Question Answer sendiri seperti, *Question Classes, Question Processing, Data Sources untuk QA, Answer Extraction, Answer Formulation, Real time Question Answering, Multilingual question answering, Interactive QA, Advance reasoning QA, Information Clustering for QA, User Profiling for QA*.

3. Cross-Domain Semantic Web Model for Understanding Multilingual Natural Language Queries: English/Arabic Health/Food Domain Use Case [9] . Tujuan dari penelitian ini adalah mengembangkan *Natural Language Query* yang mampu menggunakan *natural language question* dengan menggunakan bahasa Arab atau disebut Multilingual. Penelitian ini berfokus pada *Health/Food Domain Use Case* sebagai ontologinya. Pendekatan yang digunakan dalam pengembangannya mencakup *Query Prepocessing, Query Mapping, SPARQL Query Generator* dan yang terakhir menghasilkan sebuah output yaitu SPARQL query yang digunakan untuk mengembalikan natural language question menjadi jawaban. Penelitian ini mendorong bagi pengembangan *natural language question* dengan menggunakan multilanguage, oleh karena itu pada penelitian ini membangun sebuah *interface* yang mampu melakukan *translate* terhadap bahasa Arab untuk melakukan *information retrieval* terhadap data source yang memiliki bahasa Inggris.

2.2 Dasar Teori

2.2.1 BeinWell

BeinWell merupakan aplikasi Linked Data berbasis website yang dibangun dengan mengintegrasikan 5 dataset yang berkaitan dengan kanker prostat. Dataset yang digunakan antara lain adalah Da-



Gambar 2.1: Linking Data

taset, Cancer Gamadb, SNPedia, Comparative Toxigenomics Database (CTD), Haz-Map, Collaborative on Health and the Environment (CHE). Kelima Dataset ini dihubungkan dengan Silk Machine menggunakan XML untuk menggabungkan beberapa *file* dataset tersebut dengan menggunakan kriteria seperti yang ditunjukkan pada Gambar 2.1

Berdasarkan pada Gambar 2.1, dataset yang dapat diintegrasikan adalah Dataset SNPedia dengan Cancer Gamadb, CTD dengan Cancer Gamadb, dan CHE dengan Haz-Map

Gambar 2.2: Diagram Vocabulary BeinWell

2.2.2 Stanford POS Tagger

Part-Of-Speech Tagger (POS Tagger) merupakan sebuah sistem yang dapat membaca sekumpulan text dalam sebuah bahasa tertentu dan mengidentifikasi serta memberikan label (part of speech) pada setiap kata (dan atau token)[7] . Label tersebut merupakan jenis-jenis kata seperti, kata benda (noun), kata kerja (verb), kata sifat (adjective), dll. POS Tagging sendiri bertujuan untuk mengenai jenis atau tipe dari sebuah kata di dalam struktur kalimat. POS Tagging sendiri sudah mampu mengenali bahasa-bahasa baik, inggris, indonesia bahkan bahasa jepang. Berikut ini merupakan salah satu bentuk tagging yang dilakukan oleh Stanford POS Tagger:

This	is	a	sample	sentence
(DT)	(VBZ)	(DT)	(NN)	(NN)

Salah satu layanan POS Tagger yang terkenal dan sering digunakan adalah Stanford POS Tagger. Stanford POS Tagger juga merupakan salah satu bagian dari Stanford CoreNLP yang menyediakan layanan untuk menjalankan tugas-tugas dalam Natural Language Processing.

Stanford POS Tagger dibuat oleh Kristina Toutanova dan Christoper D. Maning pada tahun 2001 [7]. Sistem POS Tagger ini berjalan di atas JVM yang membutuhkan Java versi 1.8+ untuk mampu berjalan. Stanford POS Tagger sudah menyediakan beberapa model yang dapat digunakan untuk kebutuhan POS Tagging umum. Customisasi terhadap model juga sangat dimungkinkan bila ingin melakukan POS Tagging terhadap bahasa tertentu. Berdasarkan [6] tag atau label yang digunakan pada Stanford POS Tagging terdiri dari 36 tagging atau label yang ditunjukkan pada tabel 2.1

Tabel 2.1: Daftar POS Tagging Tag

No	Tag	Deskripsi
1	CC	Coordinating conjunction
2	CD	Cardinal number
3	DT	Determiner
4	EX	Existential there
5	FW	Foreign word
6	IN	Preposition or subordinating conjunction
7	JJ	Adjective
8	JJR	Adjective, comparative
9	JJS	Adjective, superlative
10	LS	List item marker
11	MD	Modal
12	NN	Noun, singular or mass
13	NNS	Noun, plural
14	NNP	Proper noun, singular
15	NNPS	Proper noun, plural
16	PDT	Predeterminer
17	POS	Possessive ending
18	PRP	Personal pronoun
19	PRP$	Possessive pronoun
20	RB	Adverb
21	RBR	Adverb, comparative
22	RBS	Adverb, superlative
23	RP	Particle
24	SYM	Symbol
25	TO	to
26	UH	Interjection
27	VB	Verb, base form
28	VBD	Verb, past tense
29	VBG	Verb, gerund or present participle

30	VBN	Verb, past participle
31	VBP	Verb, non-3rd person singular present
32	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34	WP	Wh-pronoun
35	WP$	Possessive wh-pronoun
36	WRB	Wh-adverb

2.2.3 Question Answering System

Question Answer Systtem merupakan sebuah sistem atau aplikasi yang menghasilkan sebuah jawaban dari pertanyaan yang ditanyakan oleh user dalam bahasa natural. Pada awal pengembangannya QAS dikembangkan pada domain area terbatas dan memiliki kemampuan yang terbatas. Fokus QAS pada pengembangan skearang adalah berfokus pada tipe pertanyaan yang secara umum ditanyakan oleh user, karakteristik dari data sources dan bentuk-bentuk dari jawaban tepat yang dihasilkan. Riset terhadap QAS sendiri bermula sejak tahun 1960 dan sejak saat itu, sejumlah QAS telah dikembangkan [11].

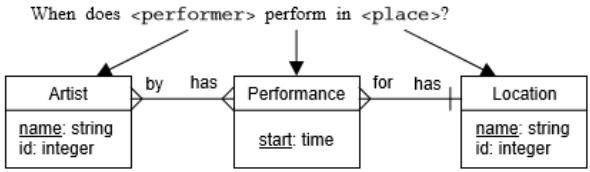
2.2.4 Template Based Approach

Template Based Approach merupakan salah satu metode yang termasuk ke dalam Pattern Matching Approach. Sehingga pendekatan ini saling terkait satu sama lain. Pattern Matching merupakan pendekatan yang mengatasi serta mendukung pola teks menggunakan pemrosesan komputer. Sedangkan pendekatan Template Based merupakan pendekatan yang mendayagunakan preformatted pattern atau pola dari sebuah question atau pertanyaan natural dari

user [12]. Menurut Eriks Sneiders, question template pada Question Answering System merupakan sebuah FAQ yang dinamis, terparameter yang merupakan lawan dari FAQ tradisional yang statis [14]. Sebuah question template adalah merupakan question dengan entity slots – tempat kosong untuk instance dari data yang merepresentasikan konsep utama dari sebuah question atau pertanyaan [14]. Contoh dari question template ditunjukkan sebagai berikut:

When does ¡performer¿ perform in ¡place¿?

Dari contoh di atas, *¡performer¿* dan *¡place¿* merupakan entity slots. Jika slot tersebut diisi dengan instance dari sebuah data yang merupakan kepemilikan dari konsep utamanya maka didapatkan pertanyaan atau question seperti : "When does Depeche Mode perform in Globen?". Entity slot yang merupakan question template answer dibuat dengan bantuan database query maupun indexing database. Entity Slots dalam question template terikat pada konsep, atau entities, dalam sebuah model konsep selama template tersebut mengekspresikan hubungan antara konsep [14]. Pada gambar 2.3 ditunjukkan contoh pengikatan sebuah question template pada sebuah atau sekeping dari sebuah konsep model [14].



Gambar 2.3: *Entity Slots* pada *question template* [14]

Satu question template meliputin sejumlah instance data yang besar yang menyinggung pada entity slots nya masing-masing. Secara

garis besar, proses dari question template hingga menghasilkan sebuah jawaban dapat dijelaskan sebagai berikut:

- mengambil instance data yang relevan dengan user question
- mengambil question template yang cocok dengan user question
- mengkombinasikan instance data yang terambil dan question template dan membuat satu atau beberapa interpretasi dari pertanyaan asli. User memilih interpretasi yang diharapkan dan question assistant menjawabnya [14].

2.2.5 Semantic Web

Sebuah fungsi tambahan dari sebuah web dimana memberikan cara yang lebih mudah untuk menemukan, berbagi, menggunakan kembali, dan menggabungkan informasi. Kemampuan ini dibentuk dengan menggabungkan kemampuan teknologi XML untuk membentuk tagging schemes dan RDF's (Resource Descripton Framework) sebagai pendekatan fleksible yang mewakili data. Semantic web menyediakan format umum untuk pertukaran data. Selain itu Semantic web juga menyediakan bahasa umum untuk merekam bagaimana data berelasi dengan obyek-obyek dunia nyata, memungkinkan orang atau sebuah mesin memulai pada satu database kemudian berhubungan degan database lain dan terkonseksi satu sama lain [**?**]. Teknologi pendukung dari Semantic Web berupa hal-hal yang akan dijelaskan sebagai berikut:

2.2.6 RDF Schema

RDF pertama kali diperkenalkan pada tahun 1998 dan kini telah menjadi sebuah standard untuk pertukaran sebuah data pada web.

Pada masa kini, sejumlah data yang masif telah dikonversikan ke dalam bentuk RDF dan dipublikasikan secara terbuka. RDF sendiri merupakan sebuah model standard untuk pertukaran data pada sebuah web. RDF memiliki fitur yang dapat melakukan data merging meskipun kedua data schema saling berbeda RDF menyediakan cara untuk menjelaskan sumber dari metadata dengan sebuah triple (subject, predicat, object).

RDFS (Resource Description Framework Schema) merupakan pengembangan atau perluasan dari RDF Vocabulary [8]. RDF Schema menyediakan sebuah vocabulary data modeling untuk RDF Data. RDFS sendiri merupakan schema language yang paling populer digunakan pada teknologi Semantic Web. RDFS mendefinisikan terminologi yang akan digunakan dalam RDF dan memberikan arti khusus kepadanya [8] . RDF Schema memiliki beberapa komponen di mana antaranya adalah, class, property dan instance [8]. Class mendefinisikan sekumpulan individu yang harus bersama karena mereka berbagi property yang sama. Class juga memiliki hierarki seperti subClassOf. Class biasanya dapat diidentifikasi oleh sebuah IRI dan mungkin dideskripsikan menggunakan RDF properties. Property merupakan cara untuk memberikan atau menyatakan sebuah hubungan antara individu satu dengan individu lain, contoh property adalah seperti hasInfected, hasHospitalized, hasDoctor. Property juga memiliki bentuk hierarki seperti subPropertyOf. Spesifikasi ini ditujukan untuk menyatakan bahwa sebuah property merupakan subproperty dari yang lain.

2.2.7 Ontology

Ontology merupakan arti serta penjelasan mengenai sebuah object, property dari sebuah object dan relasi antar object tersebut di sebuah knowledge database. Menurut [11], ”Ontology merupakan

spesifikasi formal dari sebuah konseptual yang diterima"

Ketika berbicara dalam cakupan Artificial Intelligence, ontology memiliki dua arti. Yang pertama adalah bahwa ontology sebagai representasi vocabulary ditujukan pada suatu pembahasan mengenai sebuah database tertentu. Dan pemahaman yang lebih jauhnya, secara umum pada semua bidang keilmuan sains di dunia, dapat menggunakan ontology untuk menghubungkan informasi dengan sistem. Ontology ini dapat dihubungkan ke beberapa aplikasi, database, melalui sebuah domain. Domain nya sendiri dapat berupa area dari sebuah knowledge seperti perlakuan atau subjek area tertentu.

Ontology sendiri terdiri dari class dan properties. Class merupakan sekumpulan dari individu yang mendeskripsikan bagian dari dunia. Relationship merupakan sebuah hubungan antara perihal seperti disjoinWith, union, intersection, dan lain-lain. Properties merupakan sebuah atribut deskripsi antara anggota dalam sebuah class. Properties dapat dibagi menjadi dua object properties and datatype properties. Object Properties merupakan hubungan antara sebuah objek dengan objek yang lainnya, sedangkan properties datatype adalah hubungan antara sebuah object dengan nilai dari sebuah datatype[26].

Linked Data

Linked data merupakan salah satu bagian dari pembangunan web semantik. Linked data adalah sebuah pendakatan dimana menghubungkan dan membagikan data pada web. Dengan linked data sebuah website yang memiliki padanan yang sama bisa dihubungkan satu sama lain dengan menggunakan semantic queries. Sebagai contoh apabila kita ingin mendapatkan deskripsi kota surabaya, de-

ngan menghubungkan dengan dbpedia kita tidak perlu menuliskannya lagi.
Kriteria-kriteria yang terdapat data yang dapat dihubungkan adalah sebagai berikut:

- Tersedia di internet
- Memiliki struktur data yang dapat dimengerti oleh mesin
- Tersedia dalam format non-proprietary
- Menggunakan standar dari W3C untuk open data
- Terhubung dengan sumber data lainnya di internet

2.2.8 RDF

Resource Description Framework (RDF) adalah kerangka untun mengangkan informasi dari sumber-sumber data. Sumber-sumber tersebut dapat berupa apapun, termasuk dokumen, orang, benda fisik, dan konsep-konsep abstrak. RDF ini muncul saat ini dimana Web perlu di proses oleh aplikasi, bukan hanya ditampilkan kepada orang. RDF menyediakan framework umum untuk menginformasikan data sehingga dapat dilakukan pertukaran data antar apliaksi tanpa kehilangan makna [**?**].

RDF data model mirip dengan model konseptual sederhana seperti *entity relationship model* atau *class diagram*, namun paada RDF didasarkan pada pembuatan model berdasrkan pernyataan tentang sumber daya / resources (pada web) ke dalam bentuk subject-predicate-obyek. Bentuk ini dikenal dengan nama triples pada terminologi RDF. Subyek menunjukkan sumber daya / resources, predikat menunjukkan ciri-ciri atau aspek sumber daya dan menghubungkan antara subyek dan obyek [**?**]. Untuk lebih jelasnya dapat dilihat ilustrasi di bawah ini:

Alvin	menderita	kanker prostat
(subyek)	(predikat)	(obyek)

Subyek merupakan suatu hal yang dideskripsikan. Sedangkan obyek merupakan data berupa angka, string, tanggal, ataupun URI dari suatu hal atau benda lain yang memiliki hubungan dengan subjek. Predikat merupakan merupakan suatu URI yang digunakan untuk mendeskripsikan hubungan antara subjek dengan objek. URI dari predikat diambil dari vocabularies, suatu kumpulan URI yang dapat digunakan untuk merepresentasikan informasi terkait bidang tertentu [**?**]. RDF triples memiliki dua tipe, sebagai berikut:

- Literal Triples, merupakan triples dengan RDF literal berupa string, angka, atau tanggal sebagai objek. Literal triples digunakan untuk mendeskripsikan sifat / properti dari suatu hal / data.
- RDF Links, merepresentasikan hubungan antara dua sumber data. RDF links terdiri dari tiga referensi URI. URI yang digunakan pada subjek dan objek untuk mengidentifikasi sumber data yang saling terkait, serta URI pada predikat untuk mendefinisikan keterkaitan antar data

2.2.9 SPARQL

SPARQL merupakan definisi standard Query Language dan sebuah protocol untuk akses data menggunakan RDF Data model dari Semantic Web. SPARQL hanya dapat bekerja pada semua data source yang dapat dipetakan menjadi RDF. SPARQL dapat melakukan tugas-tugas seperti, mengambil sebuah nilai dari data terstruktur atau semi terstruktur, melakukan eksplorasi dengan melakukan query hubunganyang belum diketahui, mengubah RDF Data dari satu vocabulary ke yang lainnya[]. Penjelasan tersebut mengarahkan

bahwa SPARQL merupakan RDF Query Language untuk mengambil serta memanipulasi sebuah data dalam bentuk Resource Description (RDF). Hasil dari sebuah SPARQL Query dapat direnderke dalam beberapa format seperti, XML, JSON, RDF dan HTML. SPARQL dijalankan dengan menggunakan pola triple atau subject, predicate dan object dari RDF Data.

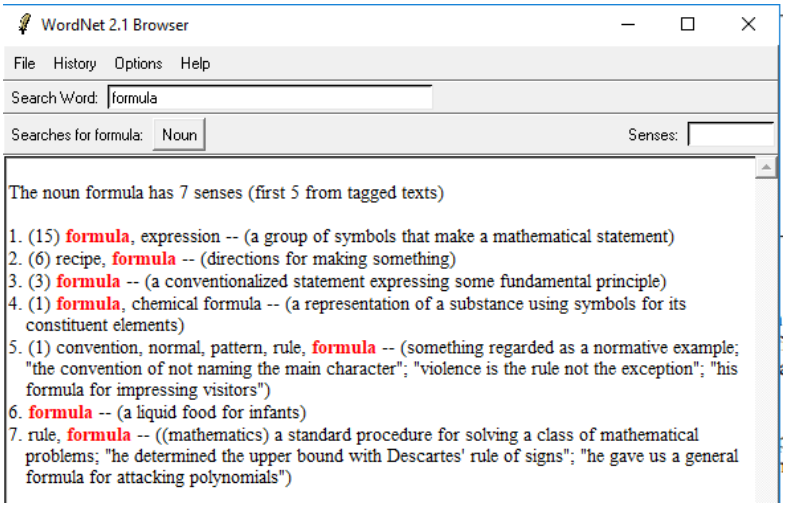
2.2.10 Wordnet

Wordnet merupakan sebuah lexical database dari bahasa Inggris atau bisa disebut lexicon. Lexicon ini mencakup, *nouns, verbs, adjective, adverbs* yang dikelompokkan menjadi suatu kumpulan *cognitive synonyms* atau disebut *synsets* [1]. Kumpulan sinonim ini dihubungan dengan makna dari sebuah *conceptual-semantic* dan *lexical relations* nya [1]. Hal ini menghasilkan sekumpulan kata yang memiliki makna semantik dan konsepnya masing-masing yang bisa dioperasikan melalui browser [1]. Wordnet sendiri bersifat *free* dan *opensource* sehingga lexicon nya sendiri dapat dikembangkan lebih lanjut. Wordnet sendiri sangat cocok digunakan untuk membuat *tools computational linguistics* dan *natural language processing*.

Pada gambar 2.4 merupakan contoh synset untuk pencarian kata *formula*

2.2.11 Levenshtein Distance

Levenshtein Distance (LD) merupakan sebuah ukuran dari kemiripan atau kecocokan antar dua *String*, di mana kita akan mengacu sebagai sumber atau disebut String (s) dan target atau disebut String t [5]. *Distance* yang dimaksud adalah jumlah dari penghapusan (*deletion*), penambahan (*insertions*) atau penggantian (*subs-*



Gambar 2.4: Hasil pencarian synsets untuk kata ”formula” pada Wordnet Browser

titution) yang dibutuhkan untuk merubah String s menjadi String t [5]. Contohnya adalah sebagai berikut:

- Jika s adalah ”test” dan t adalah ”test”, maka LD(s,t) = 0, karena tidak ada transformasi yang dibutuhkan, karena s = t
- Jika s adalah ”test” dan t adalah ”tent”, maka LD(s,t) = 1, karena dibutuhkan 1 substitusi untuk membuat s = t, sehingga cost dari Levenshtein adalah 1

Secara umum, algoritma dari Levenshtein Distance sendiri berdasarkan [5] dapat dijelaskan dari tabel 2.2 berikut:

Tabel 2.2: Pseudocode Levenshtein Distance

Step	Description

1	Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns.
2	Initialize the first row to 0..n. Initialize the first column to 0..m.
3	Examine each character of s (i from 1 to n).
4	Examine each character of t (j from 1 to m).
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.
6	Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
7	After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m]

2.2.12 Apache Jena

Apache Jena merupakan sebuah RDF API yang ditulis menggunakan Java yang juga merupakan sebuah framework untuk membangun aplikasi berbasis Semantic Web [3]. Jena menyediakan beberapa environment pemrograman untuk RDF, RDFS dan OWL, SPARQL, GRDDL serta termasuk rule-based inference engine[3]. Apache Jena memiliki beberapa komponen pendukung, di antaranya TDB, SDB dan Joseki[3]. TDB merupakan SPARQL Query Layer yang menyediakan penyimpanan non-transaction yang scalable serta ringan[3]. SDB merupakan sebuah database bagi SPARQL untuk Jena. SDB menyediakan fungsi-fungsi seperti load ba-

lancing, security, clustering, back dan administration[3]. SDB menyediakan RDF Triple Store dengan SPARQL Interface. Dan Joseki sendiri merupakan server untuk SPARQL berbasiskan Apache Jena[3]. Apache Jena sendiri memiliki tambahan layanan yang dapat digunakan sebagai server yaitu Jena-Fuseki, sebuah server SPARQL sekaligus berjalan sebagai TDB (Triple Database) dari sebuah RDF. Apache Fuseki sendiri merupakan sebuah access layer terhadap ontology atau RDF Schema. Hal ini dapat memberikan kita kemampuan untuk menyimpan RDF Data serta mengakses RDF Data tersebut melalui HTTP [2]

2.2.13 Telegram Bot

Bot API Telegram merupakan sebuah layanan interface berbasis HTTP yang dibuat untuk developer. Bot merupakan aplikasi pihak ketiga (*third party application*) yang berjalan pada platform Telegram. Pengguna dapat berinteraksi dengan bot melalui pengiriman pesan, *commands* dan *inline request*. Developer mengelola bot dengan *HTTP Request* dan Bot API.

Pada khususnya, Telegram Bot merupakan *special account* yang tidak membutuhkan sebuah nomor telepon untuk melakukan setting awal. *End User* berinteraksi dengan 2 cara, di antaranya :

- Mengirimkan pesan dan *commands* ke bot dengan memulai chat dengan bot atau menambahkan bot kepada group.
- Mengirimkan *request* secara langsung dari *input field* dengan mengetikkan *username* bot dan query permintaan. Hal ini mengizinkan pengiriman konten dari *inline bots* secara langsung ke chat manapun, *group* atau *channel*

Pesan, *commands* dan *request* yang dikirimkan oleh user akan langsung diteruskan ke software yang sedang berjalan pada server. Se-

rver perantara telegram akan mengelola semua enkripsi dan komunikasi dengan Telegram API untuk developer. Developer berkomunikasi dengan server ini melalui sebuah interface HTTPS yang sederhana yang dapat menawarkan Telegram API yang sederhana.

Halaman ini sengaja dikosongkan

BAB 3

METODOLOGI

Pada bab metodologi akan menjelaskan bagaimana langkah pengerjaan tugas akhir dengan disertakan deskripsi dari setiap penjelasan untuk masing-masing tahapan beserta jadwal kegiatan pengerjaan tugas akhir.

3.1 Tahapan pengerjaan tugas akhir

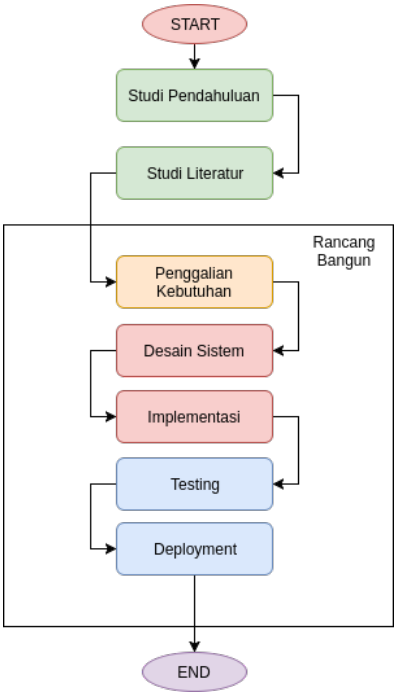
Tahapan pelaksanaan tugas akhir yang menjelaskan mengenai proses pengerjaan pada gambar 3.1

3.1.1 Studi Pendahuluan

Pada studi pendahuluan dilakukan penelitian mengenaipenelitian sebelumnya yang terkait Natural Language Question yang melibatkan RDF Data serta pendekatan-pendekatan yangdigunakan pada studi atau penelitian sebelumnya dalam membangun Natural Language Question atau Question Answering System. Dari studi pendahuluan juga mendapatkan permasalahan seputar issue atau trend yang ada pada QAS yang kemudianakan digunakan dalam membangun pendahuluan tugas akhir.

3.1.2 Studi literatur

Studi literatur yang dilakukan akan mencari mengenai toolsserta algoritma yang digunakan dalam membangun QuestionAnswering



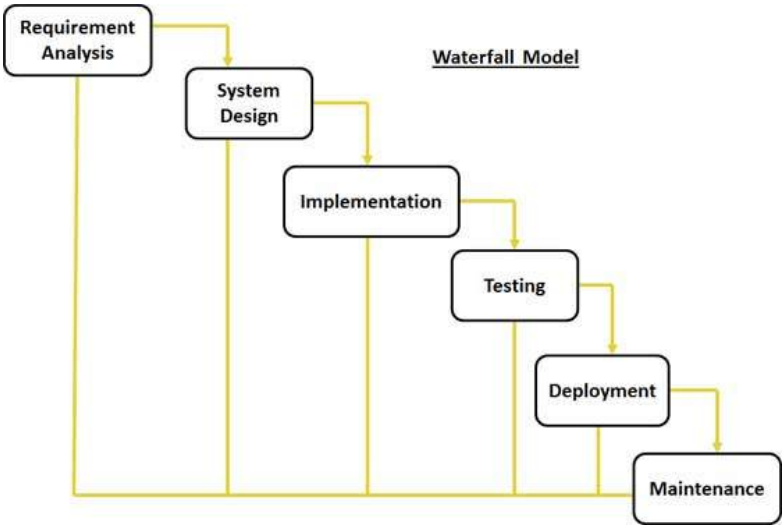
Gambar 3.1: Alur Pengerjaan Tugas Akhir

System terhadap RDF data. Hal-hal yang akan digaliterkait studi literatur adalah sebagai berikut:

- Penelitian terdahulu Question Answering System
- Framework yang membantu dalam melakukan parsing terhadap Natural Language
- Algoritma serta metode yang akan dilakukan pada pengerjaan tugas akhir

3.1.3 Rancang bangun perangkat lunak

Tahapan rancang bangun perangkat lunak pada penelitiaini akan menggunakan metode SDLC *Waterfall*. SDL*Waterfall* adalah serangkaian proses pengembangaaplikasi yang mana alur pengembangannya dimulai daratas menuju ke bawah melalui proses-prosesnya [**?**].
Proses pada model waterfall seperti ditunjukkan pada gambar 3.2 adalah sebagai berikut:



Gambar 3.2: Alur SDLC Waterfall [**?**]

1. Penggalian kebutuhan
   Pada tahap ini dilakukan penggalian kebutuhan yang akan dikembangkan ke dalam perangkat lunak. Nantinya pengguna dan pengembang aplikasi dipertemukan untuk melakukan penggalian kebutuhan perangkat lunak. Dalam penelitian ini penggalian kebutuhan akan menghasilkan *functional require-*

ment dan *non fuctional requiremment*.

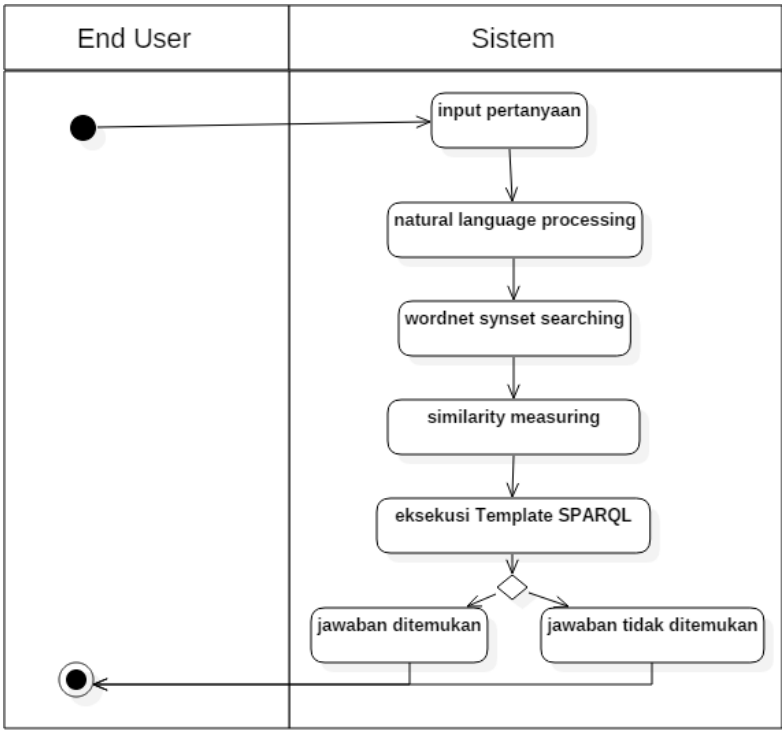
2. Desain sistem
   Pada tahap ini akan dilakukan pengkajian awal terhadap spesifikasi kebutuhan yang telah ditentukan pada tahap sebelumnya. Desain sistem nantinya akan membantu mendefinisikan kerangka arsitektur sistem secara keseluruhan. Dalam penelitian ini dilakukan desain sistem melalui tiga tahap.
   - Merancang template based query berdasarkan question type. Pada proses ini merupakan proses pembuatan template dasar SPARQL. Template SPARQL ini yang nanti akan menjadi kunci dalam proses konversi *natural language question* ke *SPARQL Query*
   - Menyiapkan RDF data prostate cancer dalam JSON-LD. Pada proses ini, RDF data dari dataset terkait dalam bentuk Turtle (.ttl) akan diparsing terlebih dahulu ke dalam bentuk JSON-LD. Hal ini dilakukan agar data tersebut dapat dilakukan indexing menggunakan Apache Lucene Solr
   - Merancang proses ekstraksi yang memanfaatkan teknologi NLP untuk mendapatkan keyword yang tepat pada pertanyaan. Proses ekstraksi melibatkan teknologi NLP seperti
     - Proses kerja dan peran POS Tagging dalam proses ekstraksi
     - Proses kerja dan peran Wordnet dalam proses analisa dan peningkatan pemahaman semantik pertanyaan
     - Proses kerja dan peran algoritma *Levenshtein Distance* untuk mendapatkan RDF Property yang tepat

     Proses dan fungsi kerja dalam NLP di atas yang menjadi pioner dalam pengimplementasian proses konversi yang melibatatkan *Human Natural Language* menjadi

SPARQL Query

- Merancang skema bot, dan fungsi-fungsi yang akan menjadi *command* dasar pada Chat bot di Telegram.

3. Implementasi
   Pada tahap ini dilakukan pengerjaan pengembangan sistem konversi natural language ke SPARQL Query dengan menggunakan RDF data kanker prostat. Implementasi secara garis besar mewujudkan struktur dari arstitektur sistem pada gambar 3.3, yaitu dengan mengembangkan serta membangun tiap elemen pada arsitektur sistem tersebut. Di samping itu tahapan ini dilakukan juga dengan memastikan bahwa sistem konversi juga dikembangkan dengan mempertimbangkan functional dan non-functional requirement. Berikut adalah tahap-tahap dalam melakukan implementasi perangkat lunak
   - Membangun *Question Processing*. Pada proses ini melakukan pemrosesan pertanyaan atau *question* seperti peng-*code*-an hal-hal berikut:
     - Tokenize, proses ini merupakan proses 'pemotongan' sebuah kalimat menjadi frasa atau kata. Proses pengerjaannya menggunakan *framework* Stanford NLP
     - POS Tagging, proses ini digunakan untuk mendeteksi serta mengekstrak kata-kata dalam pertanyaan untuk didapatkan kata benda atau NN (*Nouns*) dan kata sifat atau ADJ (*Adjective*) . Kedua tipe kata ini kemudian akan digunakan untuk dideteksi mana yang Entity dan Property pada Triples
     - Wordnet Synset Extraction Word Form, proses ini merupakan proses ekstrak sinonim kata data keyword yang didapatkan pada proses POS Tagging. Proses ini ditujukan untuk memperluas pemahaman semantik terhadap sebuah kata, sehingga makna pertanyaan bisa diperluas.



Gambar 3.3: Arsitektur Sistem Question Answering System

  - Property Extraction, proses ini melibatkan algoritma Levenshtein Distance, yaitu dengan membandingkan *String* dari keyword yang sudah dikembangkan makna semantiknya dengan string list *RDF Property*. Proses ini akan mendapatkan *candidate property* yang nantinya akan digunakan pada proses *SPARQL Template*
- Membangun *SPARQL Template Answer Retrieval*
- Membangun ontology atau RDF access layer untuk melakukan eksekusi SPARQL query sehingga dapat di-

jalankan. Service yang digunakan pada proses adalah Apache Jena.

- Melakukan integrasi Question Answering System yang dibangun pada Java dengan Chat Bot pada platform Telegram

4. Uji coba
   Pada tahap ini semua unit perangkat lunak dikembangkan menjadi satu perangkat lunak yang terintegrasi. Setelah perangkat lunak terintegrasi dilakukan uji coba secara keseluruhan untuk mengetahui kesalahan atau error pada perangkat lunak. Pada pengujian perangkat lunak apakah nantinya perangkat lunak dapat berjalan sesuai dengan rancangan yang dibangun dengan mencari error ataupun bug. Aplikasi diuji menggunakan metode sebagai berikut:
   - *Integration testing*
     - Pengujian dengan menggunakan experiment case / skenario. Pengujian ini ditujukan untuk memastikan bahwa tipe-tipe pertanyaan umum seperti didukung oleh Question Answering System. Pertanyaan-pertanyaan yang akan digunakan sebagai skenario atau *experiment case* adalah sebagai berikut:
     - Akurasi jawaban yang diambil. Pengujian ini dilakukan dengan menghitung dari semua total pertanyaan yang diberikan terhadap Question Answering System dengan membandingkan dengan tingkat keberhasilan serta ketepatan QAS dalam memberikan jawaban

     - Di samping itu akan dilakukan pengujian terhadap keberhasilan dari setiap tahapan pada proses konversi, yaitu mulai dari proses POS Tagging, Synset Extraction, Levenshtein Distance dan SPARQL

Tabel 3.1: Question Test Case List

No	Question Case
1	What is allele of variance Rs1800896?
2	How much is the magnitude of variance Rs4242382?
3	What is the repute of variance Rs4242382?
4	What is the summary of variance Rs4792311?
5	What are the causes of disease Prostate Cancer?
6	What is the evidence of the causes of methyl bromide?
7	What is the symbol of chemical pyrimidine?
8	What are the interactions of chemical tetrachlorobiphenyl?
9	What is the synonyms of chemical chlorophenoxybutyric?
10	What is the formula of chemical chlorophenoxybutyric?
11	What is the category of chemical Metaldehyde?
12	What is the source of chemical Metaldehyde?
13	What is the cas number of chemical propylphthalate?
14	Who is the researcher that has published journal Nat Genet?
15	When is publication of journal Nat Genet?
16	What is the region of researcher Eeles RA found ?
17	What is the mapped gene of researcher Eeles RA found ?
18	What is the reported gene of researcher Eeles RA found ?
19	What is allele studied by researcher Schumacher?
20	What is the causes of Prostate Cancer researched in journal Nat?

Query. Selain itu pengujian dilakukan untuk memastikan jika Question Answering yang dibangun pada arsitektur Java dapat diintegrasikan dengan baik pada platform Telegram.

- *Performance Testing*
  Pengujian ini dilakukan dengan cara menghitung yang dibutuhkan pada setiap tahapan pada sistem konversi, mulai dari waktu proses POS Tagging, Synset Extra-

ction, Levenshtein Distance dan eksekusi SPARQL Query. Serta dengan menghitung waktu total yang dibutuhkan sistem konversi dalam memberi respon ketika menerima *request*.

5. *Deployment* perangkat lunak
   Setelah ujicoba secara fungsional maupun non fungsional selesai dilakukan, perangkat lunak di luncurkan (*deployed*) ke masyarakat luas.

Halaman ini sengaja dikosongkan

BAB 4

PERANCANGAN DESAIN SISTEM

Pada bab ini akan menjelaskan proses desain atau perancangan dalam pembuatan aplikasi sebagai output dari pengerjaan Tugas Akhir. Perancangan dan desain aplikasi terdiri dari beberapa tahap seperti, *requirement analysis, system design* dan *application design*.

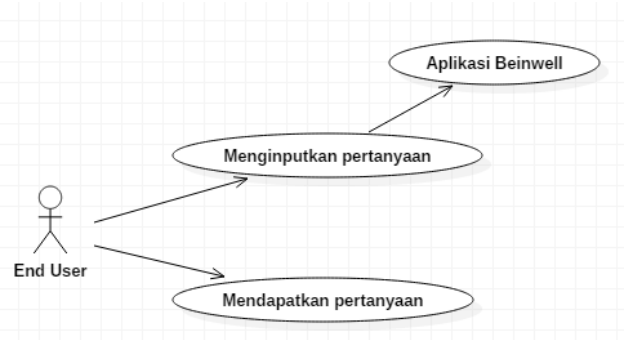
4.1 Perancangan Sistem

Pada sekumpulan tahapan ini menjelaskan mengenai desain dan perancangan sekumpulan komponen utama yang membangun sistem Transformasi Natural Language ke SPARQL. Di mana komponen ini yang akan membantu dalam proses tersebut

4.1.1 Use case Aplikasi Transformasi Natural Language ke SPARQL BeinWell

Pada tahapan ini didefinisikan mengenai rancangan alur penggunaan aplikasi BeinWell. Hal ini dijelaskan dalam use case diagram pada gambar 4.1

End User menginputkan pertanyaan dalam format bahasa inggris yang baik secara gramatikal atau tatanan bahasa, di mana kemudian pertanyaan inputan akan diproses oleh sistem untuk disusunkan SPARQL untuk memenuhi pertanyaan tersebut yang akan mengeksekusi dan mendapatkan jawaban pada dataset BeinWell. Pertanyaan yang dapat dimasukkan terbatas pada pertanyaan ”Mengapa” dan ”Bagaimana”. Keterbatasan jenis inputan pertanyaan ini di-

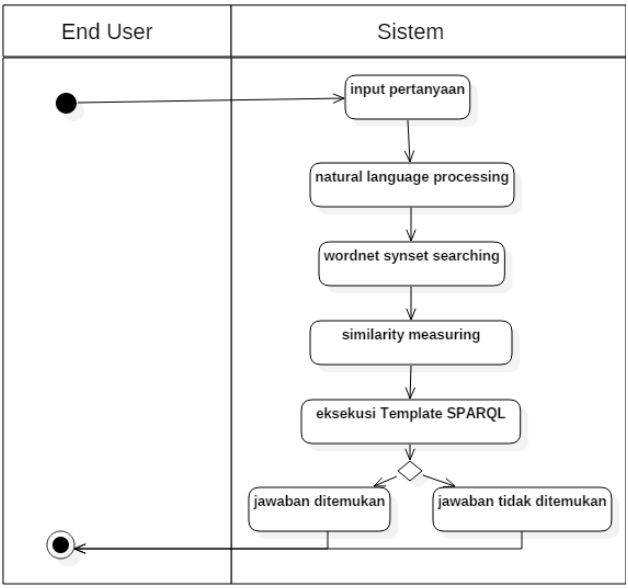


Gambar 4.1: Use Case Diagram Aplikasi BeinWell

dasarkan karena dalam pemrosesan kedua tipe pertanyaan tersebut digunakan proses reasoning yang tidak dibahas dan diadopsi pada sistem ini. Pertanyaan yang mudah dan dapat diolah pun hanya sebatas pertanyaan yang hanya memiliki pola-pola tertentu seperti bahwa nama objek yang ditanyakan harus selalu berada di akhir, seperti ”What is the symbol of chemical ATK1?”. Dari pertanyaan tersebut dapat diketahui bahwa user ingin menanyakan symbol (property) dari AKT1 (object).

4.1.2 Activity Diagram Aplikasi Transformasi Natural Language ke SPARQL BeinWell

Gambar 4.2 memperlihatkan gambaran umum proses sistem transformasi Natural Language ke SPARQL yang akan dibangun. Proses diawali dengan sebuah input pertanyaan dalam format atau bentuk Bahasa Inggris. Pertanyaan ini nantinya akan diproses pada text processing. Pertanyaan akan di-breakdown menjadi token-token. Token-token atau kata-kata ini nantinya akan dilabeli sesuai jenisnya, di mana kata atau keyword yang termasuk kata benda dan kata sifat (label N dan label J) akan diambil sebagai dasar untuk penca-



Gambar 4.2: Activity Diagram Sistem BeinWell

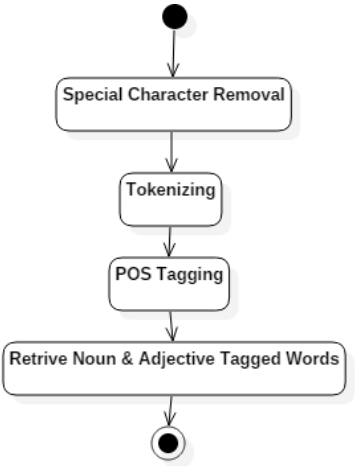
rian resource dan property. Kemudian di proses selanjutnya akan dilakukan pencarian *Synset* atau *Synonim Set* dari kata-kata yang berlabelkan N dan J. Kedua tipe kata ini akan dicarikan synonym atau padanan katanya. Tipe kata Noun atau label N akan dicarikan padanan kata bendanya, sedangkan tipe kata Adjective atau yang berlabelkan J akan ditemukan padanan katanya atau sinonimnya. Kemudian hasil padanan kata baik yang bertipe kata benda dan yang bertipe kata sifat akan dimasukkan ke dalam Array List yang merupakan kumpulan dari padanan kata-kata yang berdasarkan kata benda dan kata sifat yang terkandung dari pertanyaan pengguna. Padanan kata atau Synset ini kemudian akan dicocokan atau diukur kedekatannya menggunakan *Similarity Measurement, Levensthein Distance* antara kumpulan padanan kata dengan property yang terkandung dari RDF yang sudah dimuat ke dalam Array List berda-

sarkan dataset masing-masing. Kata yang memiliki ukuran kedekatan lebih dari atau sama dengan 70 persen maka akan diambil untuk kemudian disimpan. Di lain sisi, SPARQL Template yang sudah dibuat akan diisi dengan variable atau parameter yang sebelumnya didapatkan dari proses POS Tagging untuk mendapatkan kata benda dan Wordnet Process serta Similarity Measuring untuk mendapatkan property ini akan dieksekusikan ke Ontology Access Layer untuk mengakses data RDF untuk memenuhi pertanyaan input.

4.1.3 Activity Diagram Pemrosesan Natural Language

Pada tahap perancangan ini ditujukan untuk bagaimana sebuah pertanyaan dapat dimengerti oleh sistem, di mana entitas apa yang harus ditujukan kepada SPARQL Template untuk dapat memenuhi pertanyaan inputan. Pada proses perancangan pemrosesan Natural Language ini, digunakan proses special character removal untuk menghapus karakter khusus atau tanda baca pada pertanyaan, kemudian proses tokenizing untuk melakukan breakdown terhadap kalimat pertanyaan menjadi kumpulan kata untuk dapat dimengerti oleh sistem. Pada gambar 4.3 dijelaskan alur pada tahapan ini

Pada *special character removal* inputan seperti tanda baca, (.,?/) akan dihilangkan dari pertanyaan inputan. Kemudian pada Tokenizing satu frasa kalimat pertanyaan akan di-breakdown menjadi per word atau kata seperti, ”What is the Chemical Name of H2SO4?” maka akan menjadi ”What”, ”is”, ”the”, ”Chemical”, ”Name”, ”of”, ”H2SO4”. Kemudian dari setiap token tersebut akan dilanjutkan kepada pemberian label atau anotasi terhadap token tersebut kemudian akan diambil token yang memiliki tag yang berawalan N dan J. Tag tersebut merupakan label terhadap kata yang termasuk kata benda dan kata sifat. Sehingga jika berdasarkan kalimat pertanyaan



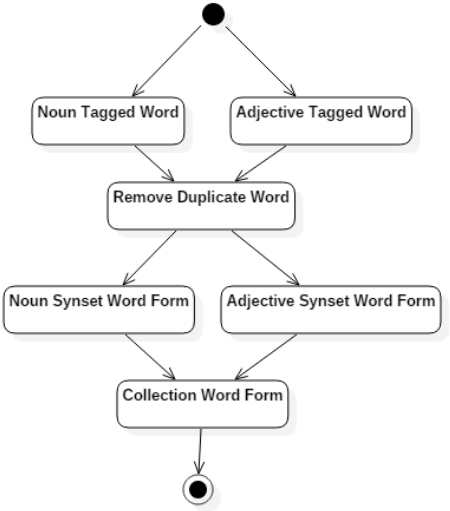
Gambar 4.3: Use Case Diagram Aplikasi BeinWell

inputan di atas maka yang akan diambil adalah, kata ”Chemical”, ”Name”, dan ”H2SO4” yang akan disimpan ke dalam *Array*.

4.1.4 Activity Diagram Wordnet Processing

Proses pencarian *Synset* atau *Synonym Set* yang ditunjukkan pada Gambar 4.4 ini bertujuan untuk mendapatkan variasi atau padanan kata dari kata-kata berlabel noun maupun adjective yang didapatkan dari pertanyaan pengguna. Pencarian padanan kata atau sinonim ini juga ditujukan untuk memperbanyak kemungkinan ditemukannya kata yang memiliki kedekatan yang cukup tinggi dengan property yang dimiliki oleh RDF. Proses ini dilakukan dengan menerima inputan terlebih dahulu dari daftar atau koleksi kata yang berlabelkan NOUN atau N dan kata yang berlabelkan ADJECTIVE atau J. Kumpulan kata ini kemudian akan dicarikan padanan katanya sesuai dengan labelnya masing-masing. Kata benda akan memiliki

padanan kata yang tergolong kata benda dan kata sifat akan memiliki padanan dari kata yang berlabelkan kata sifat. Proses pencarian sinonim ini menggunakan daftar kamus dari Wordnet. Tiap kata, baik kata berlabelkan kata benda dan kata sifat akan mendapatkan padanan kata yang masih berbentuk word, definition/term. Di sini yang digunakan hanya padanan katanya, oleh karena itu dilakukan proses ekstraksi *words* dari Synset. Setelah extraksi selesai yang dilakukan adalah menghapus atau membuang kata yang memiliki duplikat. Karena akan ditemui kata yang memiliki banyak definisi, sehingga kata tersebut akan tercetak lebih dari satu. Karena hanya dibutuhkan kata yang unik saja, maka kata yang duplikat akan dihapus. Koleksi padanan kata yang didapatkan ini kemudian akan disimpan ke dalam List.



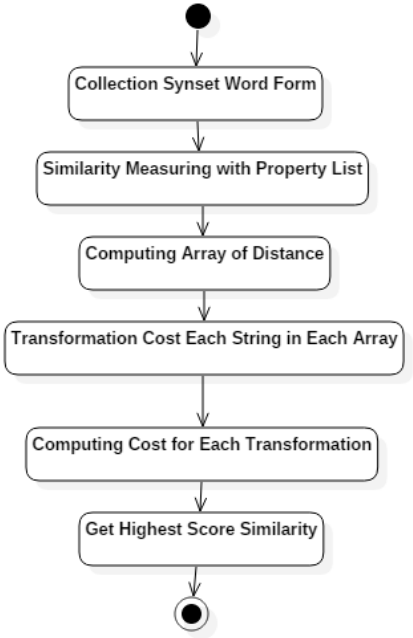
Gambar 4.4: Activity Diagram Wordnet Processing

4.1.5 Activity Diagram Similarity Measuring

Dari Gambar 4.5, ditunjukkan serangkaian proses dalam penentuan ukuran kedekatan sebuah teks antara kumpulan kata baik kata benda dan kata sifat yang sudah didapatkan di proses sebelumnya, dengan kumpulan property yang terkandung di dalam RDF Data BeinWell. Untuk metode *Textual Similarity* ini menggunakan metode *Levensthein Distance* untuk mengukur kedekatan antara String. Proses ini dilakukan dengan terlebih dahulu mendapatkan List Synset dari Wordnet yang sudah berbentuk *Word Form*, kemudian dilakukan komputasi jarak atau *distance* antar komponen terhadap komponen array lainnya. *Cost* yang digunakan pada *Levenshein* terdiri dari 3 jenis *Cost* yang digunakan, yaitu berapa langkah yang digunakan dalam mengganti huruf sehingga kedua String yang dibandingkan sama, ini disebut dengan *Cost of Replacement*, kemudian berapa langkah yang digunakan supaya String sama dengan cara penghapusan huruf atau deletion yang disebut *Cost of Deletion* dan yang terakhir adalah *Cost of Insertion*, jumlah langkah yang dibutuhkan sehingga kedua String sama dengan cara menambahkan huruf-huruf. Ketiga *cost* ini akan dikomputasikan untuk mendapatkan *cost* keseluruhan seberapa kedekatan antar komponen dari List Synset dengan List Property. Kemudian elemen yang memiliki nilai kedekatan sebesar lebih dari sama dengan 70 persen makan akan diambil untuk kemudian disimpan ke dalam List Candidate Property. Candidate Property ini yang nantinya akan dimasukkan ke dalam Template SPARQL.

4.1.6 Activity Diagram SPARQL Template Execution

Dari Gambar 4.6 proses POS Tagging, Wordnet dan Similarity Measuring telah didapatkan parameter yang dibutuhkan untuk diisikan ke dalam template SPARQL yang dibuat. Parameter ini berupa

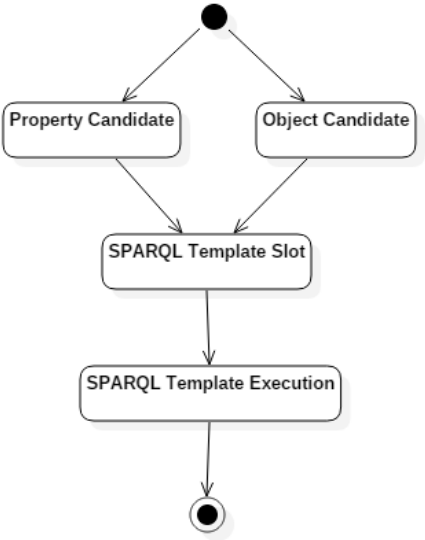


Gambar 4.5: Use Case Diagram Aplikasi BeinWell

kumpulan property serta object yang terkandung di dalam pertanyaan yang diajukan pengguna. Candidate Property akan diletakan pada slot Property dan Noun yang didapatkan dari POS tagging pada posisi terakhir (posisi terakhir dari array merupakan object yang ditanyakan) akan diletakan pada FILTER REGEX di parameter object.

Resource dan Property yang sudah didapatkan dari proses sebelumnya, kemudian akan diproses sebagai variable atau parameter di SPARQL, contoh query SPARQL ditunjukan dari gambar 4.7

Pada gambar di atas terdapat SPARQL Template yang memiliki pa-



Gambar 4.6: Activity Diagram SPARQL Template Execution

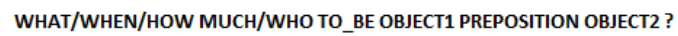
rameter atau yang bisa disebut slot. Slot ini akan diisi oleh resource dan property sesuai yang ada pada pertanyaan inputan. Di template SPARQL, pada posisi FILTER REGEX merupakan kriteria objek, slot ini diisi oleh kata benda yang didapatkan dari proses POS Tagging di awal.

Sedangkan *pattern* pertanyaan yang digunakan ditunjukkan pada Gambar 4.8:

Pola kalimat pertanyaan yang digunakan pada Question Answering System ini menggunakan pola kalimat pada Gambar 4.8, sehingga dari pola kalimat tersebut dibuat template SPARQL yang dapat mengakomodasi kebutuhan tersebut. Template SPARQL yang didukung pada Question Answer System ditunjukkan pada code 4.1

```
sparqlQuery = NS +
        "SELECT * " +
        "WHERE { " +
        "?s a <http://localhost:3030/ctd/> ." +
        "?s " + ListProperty.get(0) + "?" + ListProperty.get(0).substring(ListProperty.get(0).lastIndexOf("
        "?s " + ListProperty.get(2) + "?" + ListProperty.get(2).substring(ListProperty.get(2).lastIndexOf("
        "FILTER REGEX(?"+ ListProperty.get(2).substring(ListProperty.get(2).lastIndexOf(":") + 1) + ", \""
        "}" +
        "LIMIT 25";
```

Gambar 4.7: SPARQL Template Berparameter



Gambar 4.8: Pola Kalimat Pertanyaan

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
SELECT * WHERE {
?s rdf:type ?class: ;
   ?slotProperty1 ?predicate .
   ?slotProperty2 ?slotObject
   FILTER regex(?slotObject, "Regex_Keyword")
   } LIMIT 25
```

Kode 4.1: Template SPARQL Query

Berdasarkan *code* SPARQL 4.1, pola pertanyaan pada Gambar 4.8 dapat diakomodasi. Sehingga template ini kemudian akan diisi oleh berbagai instance data dan property dari RDF. Template SPARQL Query ini berlaku untuk pertanyaan yang hanya melakukan akses informasi pada satu dataset. Untuk mengakomodasi pertanyaan yang mengakses informasi dari lebih 1 dataset perlu template SPARQL yang berbeda, template ini ditunjukan pada *code* 4.2

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw: <http://localhost:3030/biw/>
PREFIX  ctd: <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp: <http://localhost:3030/snpedia/>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE  {
?s      rdf:type            ?class: ;
        owl:sameAs          ?same ;
        ?slotProperty1          ?predicate .
        ?same  ?slotProperty2  ?object
        FILTER regex(?object, "Regex Keyword")
        } LIMIT  25
```

Kode 4.2: SPARQL Template Akses Lebih 1 Dataset

Pada *triple* di template SPARQL pada *code* 4.2, *triple* ?s ?owl:sameAs ?same akan mengambil triple yang ekivalen dari ?subject di class dataset ?class.

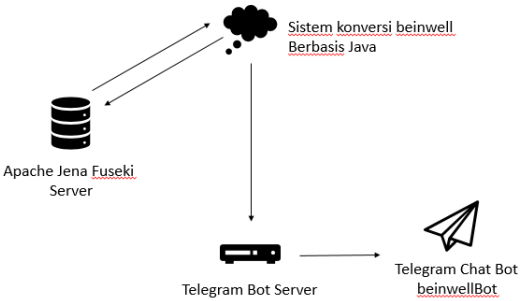
4.2 Perancangan Skema Chat Bot Telegram

Chat Bot yang akan dibangun adalah chat bot yang akan berjalan pada *platform* Telegram. Beberapa komponen yang harus dibuat pada chat bot di Telegram meliputi di antaranya:

- About - merupakan keterangan apa yang chat bot bisa lakukan
- List Command - untuk menampilkan beberapa rekomendasi *command* yang bisa dilakukan oleh chat bot

Di samping itu berikut adalah gambaran umum arsitektur Question Answering System yang diintegrasikan pada platform Telegram sesuai pada Gambar 4.9

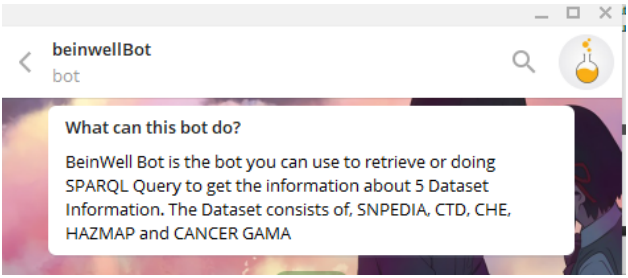
Chat Bot Telegram berdasarkan Gambar 4.9, merupakan integrasi antara Sistem Question Answering yang dibangun pada Java pada platform Telegram dengan memanfaatkan Telegram Bot API. Chat



Gambar 4.9: Arsitektur Question Answering Chat Bot Telegram

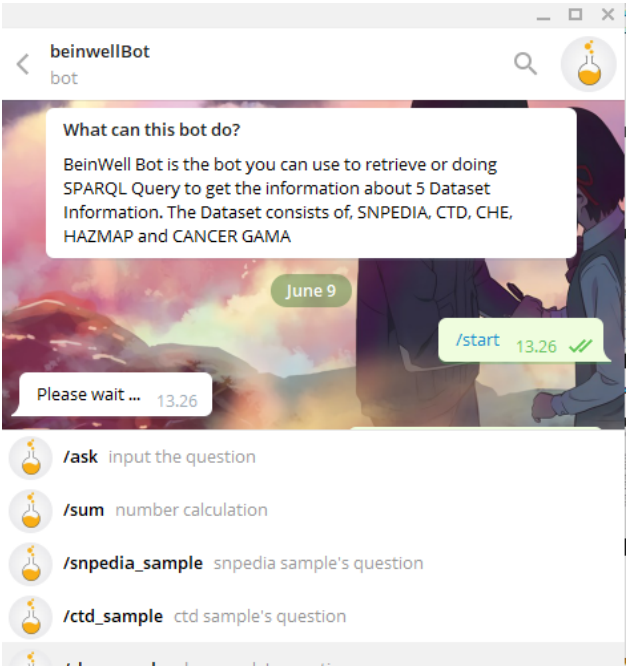
Bot yang dibangun akan menerima input pertanyaan dengan menggunakan fungsi ***commands***. Input pertanyaan akan diparse dengan menggunakan Sistem Konversi pada Java. Pertanyaan yang berhasil diparsing akan menghasilkan keyword-keyword dan property yang akan diteruskan pada SPARQL Template. SPARQL Template yang sudah berisi dengan *property* dan *keyword* yang sesuai akan diek-sekusi pada Apache Jena Fuseki, di mana kembalian yang berupa JSON akan diterima oleh Chat Bot Telegram, dan ditampilkan.

Berikut merupakan tampilan dari Chat Bot Telegram yang akan dibuat, seperti yang terlihat pada gambar 4.10



Gambar 4.10: Tampilan Welcome dari Chat Bot Telegram

Kemudian berikut merupakan tampilan rekomendasi *Command Listing* pada Chat Bot Telegram yang akan dibuat seperti pada gambar 4.11:



Gambar 4.11: Tampilan Command Listing dari Chat Bot Telegram

Di mana setelah dimasukkan term questionnya maka akan ditampilkan objek yang terkait dan memenuhi pertanyaan tersebut.

4.3 Testing Sistem

Pada tahapan testing sistem ini akan dilakukan dua macam pengujian. Yang pertama adalah pengujian secara *functional* dan penguji-

an *non functional*. Untuk pengujian secara *functional* menggunakan beberapa macam tipe pertanyaan yang dijelaskan pada tabel 3.1 . Pertanyaan-pertanyaan yang dibuat melibatkan property-property RDF pada Dataset yang ditunjukkan pada Tabel 4.1

Tabel 4.1: Daftar RDF Property

RDF Property	Class
snp:variantid	SNPEDIA
snp:allele	
snp:magnitude	
snp:repute	
snp:summary	
che:disease	CHE
che:evidence	
che:causes	
che:description	
ctd:chemicalname	CTD
ctd:symbol	
ctd:organism	
ctd:interaction	
ctd:act	
haz:chemicalname	HAZ
haz:cas	
haz:formula	
haz:majorcat	
haz:synonims	
haz:category	
haz:description	
haz:source	
pubmed:record	GAMA
pubmed:researcher	
pubmed:publication	

pubmed:journal
gama:disease
gama:region
gama:reported
gama:mapped
gama:context
gama:pvalue
gama:platform

Berdasarkan dari Tabel 4.1, RDF property yang akan digunakan sebagai pengujian adalah RDF property yang memiliki arti yang tidak ambigu dan bukan terdiri dari singkatan. RDF Property seperti, ctd:chemicalid, ctd:geneid, ctd:organismid, gama:snp, gama:snpid, gama:or, gama:95ci tidak disertakan pada pengujian dikarenakan property tersebut berupa singkatan dan tidak mengandung konteks yang dapat dipahami, kemudian property id yang tidak mengandung informasi yang bermakna karena dapat digantikan oleh property lain seperti name, sehingga property tersebut tidak disertakan untuk pengujian.

Dengan menggunakan property RDF pada Tabel 4.1, dibuat pertanyaan yang terlihat pada Tabel 4.2

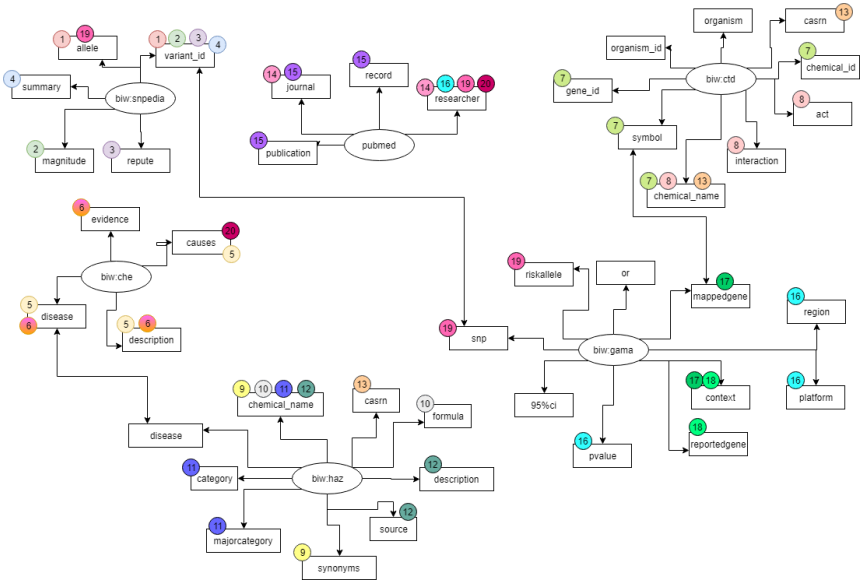
Tabel 4.2: Question Test Case

No	Question Case	Dataset
1	What is allele of variance Rs1800896?	SNPEDIA
2	How much is the magnitude of variance Rs4242382	SNPEDIA

3	What is the repute of variance Rs4242382	SNPEDIA
4	What is the summary of variance Rs4792311?	SNPEDIA
5	What are the causes of disease Prostate Cancer?	CHE
6	What is the evidence of the causes of methyl bromide?	CHE
7	What is the symbol of chemical pyrimidine?	CTD
8	What are the interactions of chemical tetrachlorobiphenyl?	CTD
9	What is the synonyms of chemical chlorophenoxybutyric?	HAZ
10	What is the formula of chemical chlorophenoxybutyric?	HAZ
11	What is the category of chemical Metaldehyde?	HAZ
12	What is the source of chemical Metaldehyde?	HAZ
13	What is the cas number of chemical propylphthalate?	CTD
14	Who is the researcher that has published journal Nat Genet?	GAMA
15	When is publication of journal Nat Genet?	GAMA
16	What is the region of researcher Eeles RA found ?	GAMA
17	What is the mapped gene of researcher Eeles RA found ?	GAMA
18	What is the reported gene of researcher Eeles RA found ?	GAMA

19	What is allele studied by researcher Schumacher?	SNPEDIA & GAMA
20	What is the causes of Prostate Cancer researched in journal Nat?	CHE & GAMA

Pemetaan pertanyaan pada arsitektur dataset ditunjukkan pada Gambar 4.12



Gambar 4.12: Distribusi pertanyaan terhadap Dataset

Di samping dengan menggunakan *question test case*, pengujian fungsional dilakukan juga dengan menilai ketepatan serta apakah sistem dapat memberikan jawaban yang diminta dengan tepat dan benar.

Pengujian secara non-fungsional akan dilakukan dengan cara menghitung waktu yang dibutuhkan pada tiap proses (proses POS Tag-

ging, Wordnet Synset Extraction dan Levenshtein Distance) untuk me-*respond request* yang diminta. Di samping itu pengujian secara fungsional juga dilakukan

BAB 5

IMPLEMENTASI

Pada bab ini akan dijelaskan terkait proses implementasi yang dilakukan pada sistem perangkat lunak yang telah dirancang sebelumnya pada Bab 4.

5.1 Lingkungan Implementasi

Pada bagian ini dibahas terkait lingkungan pengujian yang digunakan dalam implemetasi tugas akhir terkait perangkat yang digunakan baik perangkat keras maupun perangkat lunak. Tabel 5.1 yang berisikan spesifikasi perangkat keras dan perangkat lunak untuk implementasi pada tugas akhir ini.

Tabel 5.1: Spesifikasi Perangkat Keras

Perangkat	Spesifikasi
Jenis	Lenovo G41
Processor	AMD A8
RAM	4GB
Hard Disk Drive	1 TB

Kemudian untuk perangkat lunak baik yang berupa *development tools*, *modeling tools*, dan *library* yang digunakan dalam pelaksanaan proses implementasi ini dijelaskan pada Tabel 5.2. Adapun beberapa hal yang perlu diperhatikan dalam penggunaan library seperti, Stanford POS Tagging 3.7.0 yang merupakan versi terbaru tidak mendukung penggunaan SLF4J yang berada di library Apache Jena, sehingga disarankan menggunakan library Stanfod POS

Tagging versi di bawah 3.7.0

Tabel 5.2: Spesifikasi Perangkat Lunak

Nama Perangkat Lunak	Kegunaan dalam Implementasi
Xampp 5.6.30 dengan PHP 5.6.30	Webserver
Stanford POS Tagger 3.6.0	NLP Tagger
Apache Jena Fuseki 2.5.1	SPARQL Server
Apache Jena Java Library 3.3.0	Library Jena untuk Java
Wordnet 2.1	Dictionary
JAWS 1.8.0	Java API for Wordnet Searching
IntelliJ IDEA 2016.1.4	IDE untuk development Java
PhpStorm 2017.1	IDE untuk development PHP
Atom	Text Editor
Google Chrome 58	Web Browser

5.2 POS Tagging Extraction

Pada tahap ini akan dilakukan proses ekstraksi terhadap kata-kata dalam kalimat atau frasa pertanyaan atau yang dalam tugas akhir ini disebut *Natural Language Question*. Ekstraksi ini dilakukan dengan menggunakan model Stanford yang sudah tersedia secara open source yang dapat diakses di halaman website Stanford NLP [7] . Ekstraksi dilakukan untuk mengeluarkan kata-kata yang termasuk ke dalam jenis kata benda atau yang di dalam Stanford diberi label awalan N dan jenis kata sifat atau adjektiva yang diberi label awalan J dari pertanyaan yang telah diinputkan oleh pengguna. Kata-kata berlabel N dan J ini kemudian akan disimpan ke dalam sebuah List

5.2.1 Noun-Tagged Words Extraction

Pada tahap ini akan dilakukan pengambilan atau ekstraksi kata-kata benda yang terkandung di dalam sebuah kalimat atau frasa pertanyaan (*Natural Language Question*). Berikut adalah salah satu contoh pertanyaan yang dapat diajukan pada sistem ini:

> What is the chemical of symbol AKT1?

Pertanyaan tersebut akan melalui proses POS Tagging menjadi hasil di bawah ini :

> What/WP is/VBZ the/DT chemical/NN of/IN symbol/NN AKT1/NN ?/.

Dari cuplikan hasil diatas, dapat diketahui bahwa label dari kata terletak setelah tanda "/". Untuk mengambil atau melakukan ekstraksi terhadap kata benda yang berlabel NN (*Noun Singular*) dijelaskan melalui potongan code berikut:

```
//Create Tagger
MaxentTagger maxentTagger = new MaxentTagger("taggers/english-left3words-
      distsim.tagger");

//Receive User Input Question
Scanner sc = new Scanner(System.in);
System.out.println("What do you want to ask about?");
String question = sc.nextLine();

//Question Sentence is being tokenized into words
String tokenized = question.replaceAll("\\W", " ");

//Each token are being tagged
String tagged = maxentTagger.tagTokenizedString(tokenized);

//token stored to collection list
String[] collection = tagged.split(" ");
```

Kode 5.1: POS Tagging Loading Process

Pertama yang dilakukan adalah melakukan loading terhadap model POS Tagging. Model yang digunakan menggunakan model Stanford english-left3words-distsim, karena implementasi menggunakan bahasa inggris. Model yang telah dimuat akan diimplementasikan pada pertanyaan masukkan yang terlebih dahulu dilakukan proses tokenisasi, di mana kalimat pertanyaan diubah menjadi kumpulan kata atau disebut *token*. *Token* ini kemudian akan diberi label dengan menggunakan basis model english-left3words-distsim. Setiap token yang sudah diberi label kemudian disimpan ke dalam *list collection*.

```
//initiate ArrayList to store noun word labeled
ArrayList<String> noun = new ArrayList<String>();
ArrayList<String> adjective = new ArrayList<String>();

//call method to store only noun labeled word
Tagger(collection, noun, adjective);
```

Kode 5.2: Noun Labeled Word stored

Dari potongan *code* 6.4, dilakukan inisiasi dua jenis ArrayList yaitu *Noun* untuk menyimpan *noun labeled words* dan *adjective labeled words*. Menggunakan *method Tagger*, akan diekstrak kata yang mengandung label *noun* atau kata benda. Jenis label dari kata benda, memiliki awalan N, sehingga semua label yang memiliki awalan N merupakan kata yang berjenis kata benda.

Potongan dari *method Tagger* dijelaskan dari code 6.7

```
private static void Tagger(String[] Array, ArrayList ArrayListSimpan1,
      ArrayList ArrayListSimpan2){
for (String aArray : Array) {

    if (aArray.substring(aArray.lastIndexOf("_") + 1).startsWith("N")) {
        ArrayListSimpan1.add(aArray.split("_")[0]);
```

Kode 5.3: Tagger to extract noun labeled words

Dari potongan 6.7 di atas, semua *token* dilakukan substring dengan indexnya dimulai dari tanda baca *underscore* . Tanda baca tersebut

merupakan pemisah antara kata dengan labelnya masing-masing. Ilustrasinya seperti gambar 5.1 di bawah:

What_WP is_VBZ the_DT **chemical_NN** of_IN **symbol_NN AKT1_NN** ?/.

Gambar 5.1: Ilustrasi POS Tagging

Dari gambar 5.1 di atas yang berhuruf tebal merupakan jenis kata benda atau *noun*. Maka dengan methode pada code 6.7 maka mulai dari tanda baca *underscore* yang berawalan huruf N akan diambil *word form* nya tanpa tagging nya. Sehingga dari ilustrasi di atas yang terambil adalah kata *chemical, symbol, dan AKT1*. kata-kata tersebut kemudian akan disimpan pada list yang sesuai. Karena merupakan kata benda, maka disimpan pada list *noun*

5.2.2 Adjective-tagged Words Extraction

Pada tahap ini akan dilakukan pengambilan atau ekstraksi kata-kata benda yang terkandung di dalam sebuah kalimat atau frasa pertanyaan (*Natural Language Question*). Ekstrasi pada kata sifat atau adjective dimaksudkan karena property pada rdf pada umumnya mengandung kata benda atau kata sifat, oleh karena itu dilakukan ekstraksi pada kata sifat yang terkandung di dalam kalimat pertanyaan.

Potongan code berikut merupakan bagian dari *mehod Tagger* yang digunakan untuk melakukan ekstraksi kata benda pada kalimat

```
        if (aArray.substring(aArray.lastIndexOf("_") + 1).startsWith("J")) {
            ArrayListSimpan2.add(aArray.split("_")[0]);
        }
    }
```

Kode 5.4: Tagger to extract adjective labeled words

Yang berbeda dari sebelumnya adalah, jika kata benda yang diekstrak adalah semua kata yang memiliki label awalan N atau *noun*, maka jika kata sifat atau adjective yang diekstrak adalah semua kata yang mengandung label dengan awalan J. Maka jika berdasarkan pertanyaan di atas, maka tidak ada kata sifat yang terkandung di pertanyaan. Maka pada kasus ini yang akan dilanjutkan untuk diproses adalah kata benda.

5.3 Wordnet Extraction

Tahap ini adalah tahap lanjutan dari proses POS Tagging. Tahap ini menggunakan input dari kata-kata benda dan sifat yang diekstrak dari pertanyaan untuk didapatkan terminologi atau kata sinonimnya. Proses ini bertujuan untuk lebih memperluas makna ataupun kata dari pertanyaan yang dimasukkan oleh pengguna. Dengan menggunakan masukkan kata yang berhasil diekstrak oleh POS Tagging, maka kata *chemical, symbol AKT1* akan dicari kata sinonimnya.

Pertama adalah sinonim untuk kata chemical ketika dicari menggunakan wordnet adalah sebagai berikut:

The noun chemical has 1 sense (first 1 from tagged texts)

1. (10) **chemical**, chemical substance -- (produced by or used in a reaction involving changes in atoms or molecules)

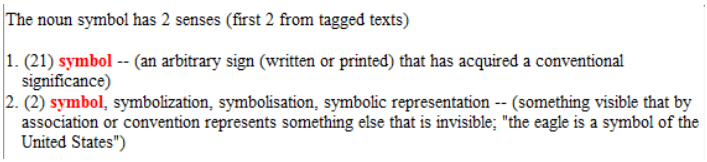
The adj chemical has 2 senses (first 2 from tagged texts)

1. (27) **chemical**, chemic -- (relating to or used in chemistry; "chemical engineer"; "chemical balance")
2. (6) **chemical** -- (of or made from or using substances produced by or used in reactions involving atomic or molecular changes; "chemical fertilizer")

Gambar 5.2: Ilustrasi Sinonim atau Synset dari kata Chemical menggunakan Worndet

Dari gambar 5.2, didapatkan bahwa sinonim dari kata chemical terdiri dari 2 jenis, yang *Synset Noun* dan *Synset Adjective*, karena kata *chemical* yang diekstrak dari POS Tagging memiliki label *Noun* atau kata benda, maka hanya perlu mencari *Synset* untuk kata chemical yang bertipe kata benda juga. Sehingga yang akan diekstrak hanya yang pertama, yaitu *chemical substance*

Kemudian untuk pencarian Synset pada kata benda *symbol*, ditunjukkan pada gambar 5.3



Gambar 5.3: Ilustrasi Sinonim atau Synset dari kata Symbol menggunakan Worndet

Pencarian di atas ditemukan sinonim kata benda untuk *symbol* adalah *symbolization, symbolisation, symbolic representation*. Dari kedua gambar di atas tersebut, diimplementasikan pada barisan code seperti pada gambar 5.5

```
//setting path for the WordNet Directory
File f = new File("WordNet\\2.1\\dict");
System.setProperty("wordnet.database.dir", "C:\\Program Files (x86)\\
    WordNet\\2.1\\dict");

//inisiasi object dictionary wordnet
WordNetDatabase wordNetDatabase = WordNetDatabase.getFileInstance();

//inisiasi list untuk menyimpan hasil wordnet extraction
ArrayList<String> keyword = new ArrayList<String>();
LinkedHashSet KeywordBersih = new LinkedHashSet();
```

Kode 5.5: Synset Extraction from Wordnet Dictionary

Barisan code 5.5 digunakan sebagai langkah awal dalam menginisiasi *Wordnet Dictionary*. Melakukan setting terhadap *PATH Wordnet*. Kemudian langkah selanjutnya adalah melakukan proses

Synset Extraction Word Form. Seperti yang telah ditunjukkan pada gambar 5.2 dan gambar 5.3 bahwa hasil ekstraksi dari *Wordnet Dictionary* akan menghasilkan daftar kata sinonim beserta terminologi penjelasannya. Dalam hal ini hanya diperlukan list kata tanpa disertai penjelasan atau definisi dari kata tersebut, oleh karena itu, tujuan ekstraksi adalah mendapatkan *Word Form* atau bentuk kata tanpa definisinya. Hal ini diimplementasikan dan dijelaskan melalui baris *code* berikut:

```
    private static ArrayList cariSinonimNoun(Synset[] synset, ArrayList simpanan
        ) {
        if (synset.length > 0) {
            ArrayList<String> listSinonim = new ArrayList<String>();
            // add elements to al, including duplicates
            LinkedHashSet CleanSet = new LinkedHashSet();
            for (int i = 0; i < synset.length; i++) {
                String[] wordForms = synset[i].getWordForms();
                for (int j = 0; j < wordForms.length; j++) {
                    listSinonim.add(wordForms[j]);
                    simpanan.add(listSinonim.get(j));

                }
                //removing duplicates
                CleanSet.addAll(simpanan);
                simpanan.clear();
                simpanan.addAll(CleanSet);

                //showing all synsets
//                for (i = 0; i < listSinonim.size(); i++) {
////                    System.out.println(listSinonim.get(i));
////                    simpanan.add(CleanSet.addAll(listSinonim));
//                    simpanan.add(listSinonim.get(i));
//
//                }
            }
        } else {
            System.err.println("No synsets exist that contain the word form");
        }
        return simpanan;
    }
```

Kode 5.6: Synset Extraction Method Word Form

Dari baris *code* 5.6, ketika berjalan *method* tersebut akan melakukan cek apakah *synset* atau sinonim set dari kata yang dicari ditemukan atau tidak sama dengan 0. Ketika kondisi diterima maka objek dengan tipe *list* akan dibuat untuk menampung list sinonim yang nantinya ditemukan. Dibuat juga list dengan tipe *Hash* yang nantinya akan digunakan sebagai kontainer untuk membersihkan synset

dari duplikasi. Proses akan diiterasi untuk mencari *Word Form* dari kata yang dicari dengan menggunakan method *getWordForm* yang berasal dari *Class* Java API Wordnet Searching. Tiap *Word Form* yang didapatkan akan disimpan ke dalam list sinonim. Kemudian karena wordnet bekerja untuk mencari kata padanan atau sinonim sehingga akan ditemui kata yang duplikat (dengan makna yang berbeda) oleh karena itu kata duplikat ini akan dihapus sehingga kata padanan yang didapatkan akan unik

```
        //Finding the Synonim of Wordnet from input
        if (!noun.isEmpty()) {
            System.out.println(noun.size() + " noun words have been found");
            System.out.println("SYNONIM WORD ARE LISTED BELOW");
            for (int i = 0; i < noun.size(); i++) {
                Synset[] synsetNoun = wordNetDatabase.getSynsets(noun.get(i),
                        SynsetType.NOUN);
                cariSinonimNoun(synsetNoun, keyword);
                keyword.addAll(noun);
//                System.out.println(keyword.get(i));
            }
        } else {
            System.out.println(noun.size() + " noun words have been found");
//            keyword.addAll(noun);
        }

        if (!adjective.isEmpty()) {
            for (int j = 0; j < adjective.size(); j++) {
                Synset[] synsetsAdjective = wordNetDatabase.getSynsets(adjective.
                        get(j), SynsetType.ADJECTIVE);
                cariSinonimNoun(synsetsAdjective, keyword);
                keyword.addAll(adjective);
//                System.out.println(keyword.get(j));
            }
        } else {
            System.out.println(adjective.size() + " adjective words have been
                found");
//            keyword.addAll(adjective);
        }

        KeywordBersih.addAll(keyword);
        keyword.clear();
        keyword.addAll(KeywordBersih);

        for (int i = 0; i < keyword.size(); i++) {
            System.out.println(keyword.get(i));
        }
```

Kode 5.7: Synset Extraction from Noun Word

Baris *code* 5.7 menjelaskan proses *Synset* untuk menemukan kata sinonim dari kumpulan kata yang sudah diekstrak pada proses POS Tagging. Pada proses ini kata yang terlabel kata benda akan melalui

proses Synset dengan tipe NOUN seperti terlihat pada baris *code* Synset[] synsetNoun = wordNetDatabase.getSynsets(aNoun, SynsetType.NOUN) dan Synset[] synsetsAdjective = wordNetDatabase.getSynsets(adjective.get(j), SynsetType.ADJECTIVE). Pada kedua baris didapatkan kumpulan kata sinonim beserta definisinya, kemudian dengan menggunakan *method* yang sudah dijelaskan pada *code* 5.6 akan didapatkan *Word Form* dari kata sinonim.

Sehingga pada tahap ini dengan melanjutkan proses sebelumya, didapatkan sebagai berikut:

***Synset* noun: chemical substance, symbolization, symbolisation, symbolic representation**

Synset Adjective: 0

5.4 Textual Similarity

Proses ini dilakukan untuk mengukur kemiripan atau kecocokan antara dua *String*. Pada pengerjaan tugas akhir ini, tahapan *Textual* Similarity yang digunakan untuk mengukur kemiripan antar dua *String* adalah menggunakan metode *Levensthein Distance*, metode ini menghitung seberapa banyak *cost* yang dikeluarkan untuk membuat String pembanding mirip dengan String patokannya. *Cost* yang digunakan pada *Levensthein Distance* adalah *cost of replacement, cost of insertion, dan cost of deletion* yang diimplementasikan dan dijelaskan pada baris *code* 5.8

```
private static int percentageOfTextMatch(String s0, String s1)
{
    // Trim and remove duplicate spaces
    int percentage = 0;
    s0 = s0.trim().replaceAll("\\s+", "␣");
    s1 = s1.trim().replaceAll("\\s+", "␣");
```

```
        percentage=(int) (100 - ((float) LevenshteinDistance(s0, s1) * 100 / (
            float) (s0.length() + s1.length())));
        return percentage;
    }

    private static int LevenshteinDistance(String s0, String s1) {

        int len0 = s0.length() + 1;
        int len1 = s1.length() + 1;
        // the array of distances
        int[] cost = new int[len0];
        int[] newcost = new int[len0];

        // initial cost of skipping prefix in String s0
        for (int i = 0; i < len0; i++)
            cost[i] = i;

        // dynamically computing the array of distances

        // transformation cost for each letter in s1
        for (int j = 1; j < len1; j++) {

            // initial cost of skipping prefix in String s1
            newcost[0] = j - 1;

            // transformation cost for each letter in s0
            for (int i = 1; i < len0; i++) {

                // matching current letters in both strings
                int match = (s0.charAt(i - 1) == s1.charAt(j - 1)) ? 0 : 1;

                // computing cost for each transformation
                int cost_replace = cost[i - 1] + match;
                int cost_insert = cost[i] + 1;
                int cost_delete = newcost[i - 1] + 1;

                // keep minimum cost
                newcost[i] = Math.min(Math.min(cost_insert, cost_delete),
                        cost_replace);
            }

            // swap cost/newcost arrays
            int[] swap = cost;
            cost = newcost;
            newcost = swap;
        }

        // the distance is the cost for transforming all letters in both strings
        return cost[len0 - 1];
    }
```

Kode 5.8: Levensthein Distance Method Code

Pada *method* ***percentageOfTextMatch*** digunakan untuk mengubah *index score* pada hasil pengukurang *Levensthein Distance* menjadi dalam ukuran presentase. Sedangkan *main method Levensthein* dijelaskan pada *method* LevenshteinDistance. Metode ini dimulai

dengan *keyword* yang sudah didapatkan pada proses *Wordnet Extraction* dibandingkan dengan daftar Property yang ada di dalam RDF Dataset Beinwell seperti code 5.10 di bawah :

```
//ArrayList of Property snpedia
ArrayList<String> snpedia = new ArrayList<String>();
snpedia.add("snp:allele");
snpedia.add("snp:variant_id");
snpedia.add("snp:magnitude");
snpedia.add("snp:repute");
snpedia.add("snp:summary");

//ArrayList of Property haz
ArrayList<String> haz = new ArrayList<String>();
haz.add("haz:chemicalname");
haz.add("haz:casrn");
haz.add("haz:formula");
haz.add("haz:majorcat");
haz.add("haz:synonims");
haz.add("haz:cat");
haz.add("haz:description");
haz.add("haz:source");

//ArrayList of Property ctd
ArrayList<String> ctd = new ArrayList<String>();
ctd.add("ctd:chemicalname");
ctd.add("ctd:chemicalid");
ctd.add("ctd:symbol");
ctd.add("ctd:geneid");
ctd.add("ctd:organism");
ctd.add("ctd:organism_id");
ctd.add("ctd:interaction");
ctd.add("ctd:act");

//ArrayList of Property che
ArrayList<String> che = new ArrayList<String>();
che.add("che:disease");
che.add("che:causes");
che.add("che:evidence");
che.add("che:description");

//ArrayList of Property gama
ArrayList<String> gama = new ArrayList<String>();
gama.add("pubmed:publication");
gama.add("pubmed:researcher");
gama.add("pubmed:date");
gama.add("pubmed:journal");
gama.add("gama:disease");
gama.add("gama:region");
gama.add("gama:reported");
gama.add("gama:mapped");
gama.add("gama:riskallele");
gama.add("gama:snp");
gama.add("gama:context");
gama.add("gama:riskallele_freq");
gama.add("gama:pvalue");
gama.add("gama:or");
gama.add("gama:gsci");
gama.add("gama:platform");

ArrayList<String> allproperty = new ArrayList<String>();
```

```
        allproperty.addAll(snpedia);
        allproperty.addAll(haz);
        allproperty.addAll(ctd);
        allproperty.addAll(che);
        allproperty.addAll(gama);
```

Kode 5.9: Property List dari RDF Dataset Beinwell dalam ArrayList

Dan berikut adalah implementasi *code* yang untuk mengukur *similarity* antar String dari keyword dan String property.

```
            int percent = 70;
        ArrayList<String> ListProperty = new ArrayList<String>();
        LinkedHashSet PropertyClean = new LinkedHashSet();

        //The process for retrive the property is using textual similarity (
             Levensthein)
        for (int i = 0; i < keyword.size(); i++) {

            for (int j = 0; j < allproperty.size(); j++) {

                int LevenPercent = percentageOfTextMatch(keyword.get(i),
                     allproperty.get(j));


                if (LevenPercent >= percent) {
//                    System.out.println(allproperty.get(j));
                    ListProperty.add(allproperty.get(j));
                }
            }
        }

        PropertyClean.addAll(ListProperty);
        ListProperty.clear();
        ListProperty.addAll(PropertyClean);

        for (int i = 0 ; i < ListProperty.size(); i++) {
            System.out.println(ListProperty.get(i));
        }
```

Kode 5.10: Property List dari RDF Dataset Beinwell dalam ArrayList

Levensthein Distance akan mengukur kedekatan atau kemiripan antara String keyword yang didapatkan pada proses *Synset Extraction* dengan daftar property dari RDF. Property dengan nilai presentasi tertinggi akan diekstrak kemudian akan disimpan ke dalam list candidate property yang nantinya akan digunakan sebagai nilai variable atau parameter pada SPARQL Template. Maka dari proses ini

output yang akan didapatkan adalah *candidate property* yang akan digunakan sebagai input atau nilai pada parameter di SPARQL Template. Jika meneruskan hasil proses sebelumnya, berdasarkan input list synset:

Input : chemical substance, symbolization, symbolisation, symbolic representation

Output : ctd:chemicalname, ctd:chemicalid dan ctd:genesymbol

5.5 SPARQL Template

Pada tahap ini, *candidate property* dan *noun* yang sudah didapatkan dari proses sebelumnya akan dimasukkan pada template SPARQL yang sudah dibuat sebelumnya. Seperti telah dijelaskan padad tahap sebelumnya, bahwa candidate property merupakan hasil dari noun extracted words dan adjective extracted words yang telah diproses pada wordnet dan *text similarity* untuk mendapatkan candidate property.

5.5.1 SNPEDIA Dataset Template

Template SPARQL untuk dataset snpedia adalah seperti yang ditunjukkan pada code 5.11 . Template SPARQL akan menerima inputan *candidate property* dan *noun*. *candidate property* akan diinputkan sebagai property, sedangkan *noun* akan diinputkan sebagai *filtering condition* pada parameter objek. Ketika SPARQL Query dieksekusi, maka kecepatan hasil query juga akan diukur.

Pada SPARQL Template untuk Dataset SNPEDIA, sebuah pertanyaan akan ditujukan pada SPARQL Template SNPEDIA ketika

pada proses Levenshtein, didapatkan bahwa pada List Candidate Property urutan pertama merupakan dataset dari SNPEDIA, sehingga dapat disimpulkan ketika sebuah pertanyaan pada objek awal mengandung konteks property SNPEDIA maka, pertanyaan tersebut akan diarahkan pada SPARQL Template SNPEDIA.

```
        final String serviceEndpoint = "http://localhost:3030/beinwellv2/query";

        String NS = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "PREFIX owl: <http://www.w3.org/2002/07/owl#>" +
                "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
                "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
                "PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
                "PREFIX snp:<http://localhost:3030/snpedia/>" +
                "PREFIX che:<http://localhost:3030/che/>" +
                "PREFIX haz:<http://localhost:3030/hazmap/>" +
                "PREFIX biw:<http://localhost:3030/biw/>" +
                "PREFIX ctd:<http://localhost:3030/ctd/>" +
                "PREFIX gama:<http://localhost:3030/gama/>" +
                "PREFIX pubmed_vocabulary:<http://bio2rdf.org/bio2rdf_vocabulary
                    :>";

        String sparqlQuery;
       if (!ListProperty.isEmpty() && !Collections.disjoint(ListProperty, snpedia
            ) && snpedia.contains(ListProperty.get(0)) && snpedia.contains(
            ListProperty.get(1)) ) {
            sparqlQuery = NS +
                    "SELECT *" +
                    "WHERE {" +
                    "?s a <http://localhost:3030/snpedia/> ." +
                    "?s " + ListProperty.get(0) + " ?p ." +
                    "?s " + ListProperty.get(1) + " ?o ." +
                    "FILTER REGEX(?o" + ", \"" + noun.get(noun.size() - 1) + "\")
                        " +
                    "}" +
                    "LIMIT 25";

            stopWatch.start();
            ExecSparql(sparqlQuery, serviceEndpoint);
            stopWatch.stop();

            long timeTaken = stopWatch.getTime();
//            System.out.println("Query Process Time: " + timeTaken / 1000 + "
    seconds");
```

Kode 5.11: SPARQL Template SNPEDIA Dataset

5.5.2 CTD Dataset Template

Berikut ini pada code 5.12 merupakan implementasi code untuk template dataset CTD. Template CTD dibuat ada 2 template, hal

ini untuk menghindari terjadinya kesalahan ketika memasukkan value yang tepat pada parameter template, hal ini dipengaruhi karena candidate property sendiri disimpan ke dalam bentuk ArrayList, sehingga urutan yang candidate property sendiri sangat berpengaruh.

Pada SPARQL Template untuk Dataset CTD, sebuah pertanyaan akan ditujukan pada SPARQL Template CTD ketika pada proses Levenshtein, didapatkan bahwa pada List Candidate Property urutan pertama merupakan dataset dari CTD, sehingga dapat disimpulkan ketika sebuah pertanyaan pada objek awal mengandung konteks property CTD maka, pertanyaan tersebut akan diarahkan pada SPARQL Template CTD. Misalkan didapatkan ketika sebuah pertanyaan masuk kemudian ditemukan keyword-keyword dan telah diperluas secara semantik. Dan pada proses Levenshtein, property yang didapatkan pada ArrayList urutan pertama, merupakan anggota bagian dari List Property CTD, maka akan dipanggil fungsi yang akan membandingkan ArrayList 1 dan ArrayList 2, jika ditemukan rdf property yang terambil dan bukan anggota dari List Property di RDF Property CTD, maka akan dihapus.

```
//Endpoint Access to BeinWell SPARQL Endpoint
final String serviceEndpoint = "http://localhost:3030/beinwellv2/query";

String NS = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
        "PREFIX owl: <http://www.w3.org/2002/07/owl#>" +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
        "PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
        "PREFIX snp:<http://localhost:3030/snpedia/>" +
        "PREFIX che:<http://localhost:3030/che/>" +
        "PREFIX haz:<http://localhost:3030/hazmap/>" +
        "PREFIX biw:<http://localhost:3030/biw/>" +
        "PREFIX ctd:<http://localhost:3030/ctd/>" +
        "PREFIX gama:<http://localhost:3030/gama/>" +
        "PREFIX pubmed_vocabulary:<http://bio2rdf.org/bio2rdf_vocabulary
            :>";

if (!ListProperty.isEmpty() && !Collections.disjoint(ListProperty, ctd)
    && ctd.contains(ListProperty.get(0)) && ListProperty.size() > 2) {
  compareArr(ListProperty, ctd);
  if (ListProperty.get(0).equals("ctd:symbol") || ListProperty.get(0).
      equals("ctd:interaction")) {
    sparqlQuery = NS +
        "SELECT * " +
        "WHERE {" +
        "?s a <http://localhost:3030/ctd/> ." +
```

```
                        "?s␣" + ListProperty.get(0) + "?p␣." +
                        "?s␣" + ListProperty.get(1) + "?o␣." +
                        "FILTER␣REGEX(?o" + ",␣\"" + noun.get(noun.size() - 1) +
                            "\")" +
                        "}" +
                        "LIMIT␣25";

                stopWatch.start();
                ExecSparql(sparqlQuery, serviceEndpoint);
                stopWatch.stop();

                long timeTaken = stopWatch.getTime();
//                System.out.println("Query Process Time: " + timeTaken / 1000 +
    " seconds");
            }
            else if (ListProperty.get(0).equals("ctd:chemicalname") ||
                 ListProperty.get(0).equals("ctd:organism") && ListProperty.size
                 () > 2) {
                compareArr(ListProperty, ctd);
                sparqlQuery = NS +
                        "SELECT␣*␣" +
                        "WHERE␣{␣" +
                        "?s␣a␣<http://localhost:3030/ctd/>␣." +
                        "?s␣" + ListProperty.get(0) + "?p␣." +
                        "?s␣" + ListProperty.get(1) + "?o␣." +
                        "FILTER␣REGEX(?o" + ",␣\"" + noun.get(noun.size() - 1) +
                            "\")" +
                        "}" +
                        "LIMIT␣25";

                stopWatch.start();
                ExecSparql(sparqlQuery, serviceEndpoint);
                stopWatch.stop();

                long timeTaken = stopWatch.getTime();
//                System.out.println("Query Process Time: " + timeTaken / 1000 +
    " seconds");
            }
```

Kode 5.12: SPARQL Template CTD Dataset

5.5.3 CHE Dataset Template

Berikut ini merupakan code implementasi dari SPARQL Template untuk dataset CHE.

Untuk sebuah pertanyaan dapat terpetakan ke SPARQL Template CHE, maka pertanyaan tersebut ketika diparse atau diproses pada tahapan Levenshtein Distance, property yang didapatkan pada urutan pertama merupakan property anggota List Property CHE. Jika

ditemukan property pada urutan kedua hingga terakhir yang bukan anggota dari himpunan List Property CHE, maka property tersebut akan dihapus, menggunakan method yang sudah dibuat yaitu, *compareArr*

```
        //Endpoint Access to BeinWell SPARQL Endpoint
        final String serviceEndpoint = "http://localhost:3030/beinwellv2/query";

        String NS = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "PREFIX owl: <http://www.w3.org/2002/07/owl#>" +
                "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
                "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
                "PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
                "PREFIX snp:<http://localhost:3030/snpedia/>" +
                "PREFIX che:<http://localhost:3030/che/>" +
                "PREFIX haz:<http://localhost:3030/hazmap/>" +
                "PREFIX biw:<http://localhost:3030/biw/>" +
                "PREFIX ctd:<http://localhost:3030/ctd/>" +
                "PREFIX gama:<http://localhost:3030/gama/>" +
                "PREFIX pubmed_vocabulary:<http://bio2rdf.org/bio2rdf_vocabulary
                    :>";


        else if (!ListProperty.isEmpty() && !Collections.disjoint(ListProperty,
              che) && che.contains(ListProperty.get(0)) && ListProperty.size() >=
               2 && che.contains(ListProperty.get(1))) {
            compareArr(ListProperty, che);
            sparqlQuery = NS +
                    "SELECT * " +
                    "WHERE {" +
                    "?s a <http://localhost:3030/che/> ." +
                    "?s " + ListProperty.get(0) + " ?p ." +
                    "?s " + ListProperty.get(1) + " ?o ." +
                    "FILTER REGEX(?o" + ", \"" + noun.get(noun.size() - 1) + "\")
                        " +
                    "}" +
                    "LIMIT 25";

            stopWatch.start();
            ExecSparql(sparqlQuery, serviceEndpoint);
            stopWatch.stop();

            long timeTaken = stopWatch.getTime();
//            System.out.println("Query Process Time: " + timeTaken / 1000 + "
    seconds");
```

Kode 5.13: SPARQL Template CHE Dataset

5.5.4 HAZ Dataset Template

Berikut ini merupakan code implementasi dari SPARQL Template untuk dataset HAZ. Sebuah pertanyaan yang akan terpetakan pa-

da SPARQL Template HAZMAP, adalah pertanyaan yang ketika diparse pada tahap Levenhstein Distance, didapatkan property pada urutan pertama di List Candidate Property adalah property di anggota List Property HAZMAP, jadi ketika didapatkan anggota property yang bukan termasuk di property HAZMAP maka akan dihapus dengan menggunakan fungsi *compareArr*

```
        //Endpoint Access to BeinWell SPARQL Endpoint
        final String serviceEndpoint = "http://localhost:3030/beinwellv2/query";

        String NS = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "PREFIX owl: <http://www.w3.org/2002/07/owl#>" +
                "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
                "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
                "PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
                "PREFIX snp:<http://localhost:3030/snpedia/>" +
                "PREFIX che:<http://localhost:3030/che/>" +
                "PREFIX haz:<http://localhost:3030/hazmap/>" +
                "PREFIX biw:<http://localhost:3030/biw/>" +
                "PREFIX ctd:<http://localhost:3030/ctd/>" +
                "PREFIX gama:<http://localhost:3030/gama/>" +
                "PREFIX pubmed_vocabulary:<http://bio2rdf.org/bio2rdf_vocabulary
                    :>";

       else if (!ListProperty.isEmpty() && !Collections.disjoint(ListProperty,
            haz) && haz.contains(ListProperty.get(0))) {
            compareArr(ListProperty, haz);
            sparqlQuery = NS +
                    "SELECT *" +
                    "WHERE {" +
                    "?s a <http://localhost:3030/hazmap/> ." +
                    "?s " + ListProperty.get(0) + "?p ." +
                    "?s " + ListProperty.get(1) + "?o ." +
                    "FILTER REGEX(?o" + ", \"" + noun.get(noun.size() - 1) + "\")
                        " +
                    "}" +
                    "LIMIT 25";

            stopWatch.start();
            ExecSparql(sparqlQuery, serviceEndpoint);
            stopWatch.stop();

            long timeTaken = stopWatch.getTime();
//            System.out.println("Query Process Time: " + timeTaken / 1000 + "
     seconds");
```

Kode 5.14: SPARQL Template HAZ Dataset

5.5.5 GAMA Dataset Template

Berikut ini merupakan implementasi code untuk SPARQL Template pada dataset GAMA, seperti yang ditunjukkan pada code 5.15. Pertanyaan yang akan terpetakan ke SPARQL Template CANCER GAMA, adalah pertanyaan yang ketika diproses akan menghasilkan sekumpulan atau List Candidate Property di mana property pada urutan pertama adalah anggota dari List RDF Property CANCER GAMA, sehingga jika ditemukan pada List Candidate Property yang bukan termasuk anggota atau subset dari List Property RDF CANCER GAMA akan dihapus, jadi dipastikan pada List Candidate Property akan memiliki hanya RDF Property yang merupakan anggota dari dataset CANCER GAMA.

```
        //Endpoint Access to BeinWell SPARQL Endpoint
        final String serviceEndpoint = "http://localhost:3030/beinwellv2/query";

        String NS = "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
                "PREFIX owl: <http://www.w3.org/2002/07/owl#>" +
                "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
                "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
                "PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
                "PREFIX snp:<http://localhost:3030/snpedia/>" +
                "PREFIX che:<http://localhost:3030/che/>" +
                "PREFIX haz:<http://localhost:3030/hazmap/>" +
                "PREFIX biw:<http://localhost:3030/biw/>" +
                "PREFIX ctd:<http://localhost:3030/ctd/>" +
                "PREFIX gama:<http://localhost:3030/gama/>" +
                "PREFIX pubmed_vocabulary:<http://bio2rdf.org/bio2rdf_vocabulary
                    :>";

            else if (!ListProperty.isEmpty() && !Collections.disjoint(
                    ListProperty, gama) && gama.contains(ListProperty.get(0))) {
                compareArr(ListProperty, gama);
                if (ListProperty.size() == 1){
                    sparqlQuery = NS +
                            "SELECT *" +
                            "WHERE {" +
                            "?s a <http://localhost:3030/gama/> ." +
                            "?s " + ListProperty.get(0) + "?" + ListProperty.get(0).
                                    substring(ListProperty.get(0).lastIndexOf(":") + 1)
                                     + " ." +
                            "}" +
                            "LIMIT 25";

                    StopWatch stopWatch = new StopWatch();
                    stopWatch.start();
                    ExecSparql(sparqlQuery, serviceEndpoint);
                    stopWatch.stop();

                    long timeTaken = stopWatch.getTime();
```

```
            System.out.println("Query Process Time: " + timeTaken / 1000 + "
                seconds");

        }

        else if (ListProperty.size() > 1 ) {
            compareArr(ListProperty, gama);
            System.out.println(ListProperty.get(0));
            System.out.println(ListProperty.get(1));
            sparqlQuery = NS +
                    "SELECT * " +
                    "WHERE {" +
                    "?s a <http://localhost:3030/gama/> ." +
                    "?s " + ListProperty.get(0) + "?" + ListProperty.get(0).
                            substring(ListProperty.get(0).lastIndexOf(":") + 1)
                             + " ." +
                    "?s " + ListProperty.get(1) + "?" + ListProperty.get(1).
                            substring(ListProperty.get(1).lastIndexOf(":") + 1)
                             + " ." +
                    "FILTER REGEX(?" + ListProperty.get(1).substring(
                            ListProperty.get(1).lastIndexOf(":") + 1) + ", \""
                            + noun.get(noun.size() - 1) + "\")" +
                    "}" +
                    "LIMIT 25";

            StopWatch stopWatch = new StopWatch();
            stopWatch.start();
            ExecSparql(sparqlQuery, serviceEndpoint);
            stopWatch.stop();

            long timeTaken = stopWatch.getTime();
            System.out.println("Query Process Time: " + timeTaken / 1000 + "
                seconds");

        }
    }
```

Kode 5.15: SPARQL Template GAMA Dataset

5.6 Building the Telegram Bot Application

Pada tahap sebelumnya merupakan tahap pembangunan dan pengembangan *core system* konversi *Natural Language Question* menjadi SPARQL Query. Agar dapat digunakan dengan mudah, maka akan dibangun *Chat Bot* yang akan dideploy pada sistem *Chat Messaging* Telegram. Chat bot ini akan terdiri dari beberapa command yaitu:

- *Commmand* ask untuk memasukkan pertanyaan yang diajuk-

an kepada chatbot
- *Command* snpedia sample untuk mengeluarkan contoh pertanyaan pada dataset snpedia
- *Command* ctd sample untuk mengeluarkan contoh pertanyaan pada dataset ctd
- *Command* che sample untuk mengeluarkan contoh pertanyaan pada dataset che
- *Command* haz sample untuk mengeluarkan contoh pertanyaan pada dataset haz
- *Command* gama sample untuk mengeluarkan contoh pertanyaan pada dataset gama

Dalam pembuatan bot berikut, pertama adalah menyiapkan environment nya, menggunakan node js, dengan memulai project dengan perintah *npm init*, yang kemudian akan menghasilkan package.json yang berisi informasi mengenai project, serta dependency yang akan digunakan. Berikut adalah konten *package.json* dari project chat bot ini, ditunjukkan pada code 5.16

```
{
  "name": "nikomania",
  "version": "1.0.0",
  "description": "beinwell bot",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "nikomata",
  "license": "ISC",
  "dependencies": {
    "node-telegram-bot-api": "^0.27.1"
    }
}
```

Kode 5.16: package.json configuration

Bagian *main* merupakan file yang akan menjadi class utama yang dieksekusi, di mana akan menjadi fungsi utama dari chat bot. Pada bagian *dependency* merupakan daftar dari library atau api yang digunakan untuk membangun chat bot. Library yang digunakan pada pembangunan chat bot ini adalah *node telegram bot api 0.27.1*.

Konfigurasi ini kemudian dieksekusi dengan perintah *npm install* akan menghasilkan struktur project baru yaitu package.json dan node modules.

```
        const TelegramBot = require('node-telegram-bot-api');
        const exec = require('child_process').exec;

        // the value below is Telegram token you receive from @BotFather
        const token = '295581169:AAHIJfsiOkOu7b6O_u1j8rXg1jGhLkh2YKE';

        // Create a bot that uses 'polling' to fetch new updates
        const bot = new TelegramBot("295581169:
             AAHIJfsiOkOu7b6O_u1j8rXg1jGhLkh2YKE", {polling: true});

        console.log('bot␣server␣started...');
```

Kode 5.17: package.json configuration

Pada code 5.18 berfungsi ketika user mengirimkan pesan kepada chat bot untuk diterima, kemudian sebagai respon ketika pesan diterima dan menunjukkan bahwa proses yang diminta sedang diproses maka dikirimkan pesan tersebut.

```
        bot.on('message', (msg) => {
         const chatId = msg.chat.id;
         bot.sendMessage(chatId, 'Please␣wait␣...␣');
        });
```

Kode 5.18: package.json configuration

Pada code 5.19 merupakan implementasi untuk command snpedia sample, ctd sample, che sample, haz sample, dan gama sample. Code ini berfungsi untuk mengirimkan kembalian berupa contoh-contoh pertanyaan yang dapat dicoba.

```
bot.on("text", (message) => {

        const child = exec('java␣-jar␣BeinWell2.jar␣"What␣is␣allele␣of␣variance␣
             Rs10090154?"',
        function (error, output, stderr){
        //console.log(output);
        hasiljson= JSON.parse(output);
        if(error !== null){
        console.log(error);
}
```

```
  if(message.text.toLowerCase().indexOf("/snpedia_sample") === 0){

      var snpedia = 'SNPEDIA sample question : What is allele of variance
           Rs138213197?';
      bot.sendMessage(message.chat.id, snpedia);

    } else if (message.text.toLowerCase().indexOf("/ctd_sample") === 0){

        var ctd = 'CTD sample question : What is the symbol of chemical
             boswellic?';
        bot.sendMessage(message.chat.id, ctd);

      } else if (message.text.toLowerCase().indexOf("/che_sample") === 0){

        var che = 'CHE sample question : What is the reason of disease Prostate
             Cancer?';
        bot.sendMessage(message.chat.id, che);

      } else if (message.text.toLowerCase().indexOf("/haz_sample") === 0){

        var haz = 'HAZ sample question : What is the formula of chemical
             chlorophenoxybutyric?';
        bot.sendMessage(message.chat.id, haz);

      } else if (message.text.toLowerCase().indexOf("/author_contact") === 0){

        var contact = 'send me email to : herjunoniko@gmail.com';
        bot.sendMessage(message.chat.id, contact);

      }
});
});
```

Kode 5.19: package.json configuration

Pada code 5.17 menjelaskan penggunaan library pada bot, token yang akan menghubungkan program chatbot dengan server telegram. Dan penggunaan child process untuk melakukan eksekusi jar file yang merupakan program utama untuk melakukan konversi *Natural Language Question* ke SPARQL Query. Chat bot ini menggunakan sistem polling, di mana bot akan berjalan secara selamanya ketika server dijalankan, hal ini juga dilakukan untuk menghindari *crash* pada bot

```
      // Matches "/ask ["Question to be asked"]"
      bot.onText(/\/ask (.+)/, (msg, match) => {

      const chatId = msg.chat.id;
      const resp = match[1]; // the captured "whatever"
      bot.sendMessage(chatId, resp);

      const child = exec('java -jar BeinWell2.jar' + ' ' + match[1], function (
           error, output, stderr){
```

```
            hasiljson= JSON.parse(output);
            if(error !== null){
            console.log(error);
}
        var length = Object.keys(hasiljson.results.bindings).length;

        //looping send message
            for (var i = 0; i < length; i++) {
                var chemical = hasiljson.results.bindings[i].p.value;
                bot.sendMessage(chatId, chemical);
        }

  });

});
```

Kode 5.20: Implementasi command ask

Code 5.20 merupakan implementasi dari command *ask*, command ini akan menerima input pertanyaan yang dimasukkan setelah command tersebut. *Function* pada code tersebut memiliki parameter msg dan match. Parameter msg berfungsi untuk menerima id chat yang dari user, sedangkan match berfungsi untuk menerima input dari user setelah command *ask*. Kemudian bot akan mengirimkan ulang pertanyaan yang dimasukkan. Di saat yang bersamaan, pertanyaan akan diproses oleh jar file yang dieksekusi oleh *child process* . Proses ini berjalan secara *async*. Hasil dari proses ini adalah output SPARQL yang telah diubah menjadi JSON, JSON ini akan di parse menjadi oject JSON untuk dapat diakses node-node JSONnya. Karena pola pertanyaan hanya menanyakan objek pertanyaannya saja maka telegram akan mengirimkan kembali nilai *object* yang didapat dari JSON

Halaman ini sengaja dikosongkan

BAB 6

HASIL DAN PEMBAHASAN

Pada bab ini akan dijelaskan hasil dan pembahasan dari proses pengujian aplikasi.

6.1 Hasil Pengujian

Pada bagian ini akan dijelaskan hasil pengujian dari *core system* yang dibangun pada arsitektur Java, serta pengujian pada Telegram Chat Bot yang meminta *request* dan menerima *respond* dari *core system*nya.

6.1.1 Pengujian Integrasi

Pada pengujian *integrasi* dilakukan dengan cara memastikan bahwa functional system yang terdiri dari POS Tagging, Semantic Extraction atau Synset Extraction Wordnet dan Levensthein Distance Measuring serta SPARQL Execution berjalan dengan baik dan terintegrasi dengan baik sebagaimana mestinya. Setelah memastikan semua tahapan proses dilalui, yang dilakukan pada pengujian functional ini adalah dengan memberika *question test case* kepada system, untuk menguji apakah system mampu menjawab pertanyaan dengan baik. Kemudian pengujian functional yang kedua adalah dengan menguji functional pada Chat bot Telegram. Functionalitas yang terdapat pada command yang diberikan agar dapat merespon dengan baik dan benar

Question Test Case 1:

Question : What is allele of variance Rs1800896?

Hasil dari POS Tagging Process adalah sebagai berikut:

```
What do you want to ask about?
What is allele of variance Rs1800896?
The following keyword have been found : allele
The following keyword have been found : variance
The following keyword have been found : Rs1800896
```

Gambar 6.1: POS Tagging Test Case 1

Dari hasil *question test case 1* didapatkan di proses POS Tag, kata yang terekstrak adalah seperti gambar 6.1. Kemudian dari kata *allele, variance* dan *Rs1800896* akan diproses sebagai input pada proses Synset Extraction. Di mana hasilnya ditunjukkan pada gambar 6.2

```
allele
allelomorph
variance
Rs1800896
discrepancy
variant
variability
variableness
```

Gambar 6.2: Hasil Synset Extraction Question Test Case 1

Dari gambar 6.2 merupakan hasil dari ekstraksi yang dilakukan dengan merujuk pada kamus Wordnet. Proses ini akan melakukan ekstraksi pada kata yang berkaitan dengan kata inputan. Di mana hasil yang di dapatkan yaitu kata-kata: *allele, allelemorph, variance, Rs1800896, discrepancy, variant, variability, variableness*

Dengan synset atau synonym sets yang sudah didapatkan akan dicari property RDF yang terkait atau memiliki kemiripan atau similar dengan Synset tersebut. Hasilnya ditunjukkan pada gambar 6.3

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
snp:allele
snp:variant_id
```

Gambar 6.3: Hasil Property Text Similarity Question Test Case 1

Kemudian property tersebut akan menjadi nilai dari parameter di SPARQL Template, sehingga slot-slot kosong pada template akan diisi oleh nilai dari property serta anggota terakhir dari ArrayList *noun*. SPARQL Query seperti ditunjukkan pada code 6.1

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE  {
?s  rdf:type        snp: ;
    snp:allele      ?p ;
    snp:variant_id  ?o
    FILTER regex(?o, "Rs1800896")  }
    LIMIT  25
```

Kode 6.1: SPARQL Query Question Test Case 1

SPARQL Query yang dieksekusi melalui Fuseki Server kemudian akan menghasilkan nilai berupa JSON seperti pada code 6.2

```
{
"head": {
  "vars": [ "s" , "p" , "o" ]
```

```
    } ,
    "results": {
      "bindings": [
        {
          "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/25" }
          ,
          "p": { "type": "literal" , "value": "(A;A)" } ,
          "o": { "type": "literal" , "value": "Rs1800896" }
        } ,
        {
          "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/26" }
          ,
          "p": { "type": "literal" , "value": "(A;G)" } ,
          "o": { "type": "literal" , "value": "Rs1800896" }
        } ,
        {
          "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/42" }
          ,
          "p": { "type": "literal" , "value": "(G;G)" } ,
          "o": { "type": "literal" , "value": "Rs1800896" }
        }
      ]
    }
  }
```


Kode 6.2: Hasil JSON SPARQL Query Test Case 1

Question Test Case 2:
Question : How much is the magnitude of variance Rs4242382?

Hasil dari POS Tagging Process pada Test Case 2 adalah sebagai berikut:

```
What do you want to ask about?
How much is the magnitude of variance Rs4242382?
The following keyword have been found : magnitude
The following keyword have been found : variance
The following keyword have been found : Rs4242382
```

Gambar 6.4: POS Tagging Test Case 2

Dari hasil pada Pos Tagging Case, dijadikan inputan pada proses Synset Extraction menghasilkan kumpulan Synset seperti pada gambar 6.5

```
magnitude
variance
Rs4242382
discrepancy
variant
variability
variableness
much
```

Gambar 6.5: Hasil Synset Extraction Question Test Case 2

Dengan synset atau synonym sets yang sudah didapatkan akan dicari property RDF yang terkait atau memiliki kemiripan atau similar dengan Synset tersebut. Hasilnya ditunjukkan pada gambar 6.6

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
snp:magnitude
snp:variant_id
```

Gambar 6.6: Hasil Property Text Similarity Question Test Case 2

Kemudian property tersebut akan menjadi nilai dari parameter di SPARQL Template, sehingga slot-slot kosong pada template akan diisi oleh nilai dari property serta anggota terakhir dari ArrayList *noun*. SPARQL Query seperti ditunjukkan pada code 6.3

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE  {
?s  rdf:type        snp: ;
    snp:magnitude   ?p ;
    snp:variant_id  ?o
    FILTER regex(?o, "Rs4242382")
    }
    LIMIT  25
```

Kode 6.3: SPARQL Query Question Test Case 2

SPARQL Query yang dieksekusi melalui Fuseki Server kemudian akan menghasilkan nilai berupa JSON seperti pada code 6.4

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/27
            " } ,
        "p": { "type": "literal" , "value": "2.0" } ,
        "o": { "type": "literal" , "value": "Rs4242382" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/28
            " } ,
        "p": { "type": "literal" , "value": "2.0" } ,
        "o": { "type": "literal" , "value": "Rs4242382" }
      }
    ]
  }
}
```

Kode 6.4: Hasil JSON SPARQL Query Test Case 2

Question Test Case 3:
Question : What is the repute of variance Rs4242382?

Hasil dari POS Tagging Process pada Test Case 3 adalah sebagai berikut:

Dari hasil Gambar 6.7 didapatkan jika keyword yang ditemukan adalah *repute, variance* dan *Rs4242382*. Dari sekumpulan keyword ini kemudian akan proses secara semantik untuk menemukan *Syn-*

```
What is the repute of variance Rs4242382?
The following keyword have been found : repute
The following keyword have been found : variance
The following keyword have been found : Rs4242382
```

Gambar 6.7: POS Tagging Test Case 3

set atau sinonim kata dari *keyword* yang ditemukan pada *POS Tagging*. Luaran yang dihasilkan ditunjukkan pada Gambar 6.8

```
repute
reputation
variance
Rs4242382
discrepancy
variant
variability
variableness
```

Gambar 6.8: Synset Extraction Test Case 3

Berdasarkan hasil *Synset Extraction* pada Gambar 6.8, langkah selanjutnya adalah diproses untuk diukur kemiripan stringnya dengan RDF Property. Hasil dari tahap ini ditunjukkan pada Gambar 6.9

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
snp:repute
snp:variant_id
```

Gambar 6.9: Hasil Property Text Similarity Question Test Case 3

Dengan *candidate property* yang didapatkan, kedua element tersebut kemudian akan menjadi nilai pada parameter yang ada pada *SPARQL Template*, sehingga akan dieksekusi sesuai pada *code* 6.5

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{
?s  rdf:type        snp: ;
    snp:repute      ?p ;
    snp:variant_id  ?o
    FILTER regex(?o, "Rs4242382")
    }
    LIMIT   25
```

Kode 6.5: SPARQL Query Question Test Case 3

Setelah SPARQL pada 6.5 dieksekusi, maka akan menghasilkan nilai JSON seperti pada *code* 6.6

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/27
            " } ,
        "p": { "type": "literal" , "value": "2.0" } ,
        "o": { "type": "literal" , "value": "Rs4242382" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/28
            " } ,
        "p": { "type": "literal" , "value": "2.0" } ,
        "o": { "type": "literal" , "value": "Rs4242382" }
      }
    ]
  }
}
```

Kode 6.6: Hasil JSON SPARQL Query Test Case 3

Question Test Case 4:
Question : What is the summary of variance Rs4792311?

Hasil dari POS Tagging Process pada Test Case 4 adalah sebagai berikut, ditunjukkan pada Gambar 6.10:

```
What do you want to ask about?
What is the summary of variance Rs4792311?
The following keyword have been found : summary
The following keyword have been found : variance
The following keyword have been found : Rs4792311
```

Gambar 6.10: POS Tagging Test Case 4

Dengan hasil *keyword* yang ditemukan dari *POS Tagging*, keyword yang didapatkan adalah *summary, variance* dan *keyword* yang terakhir merupakan *noun* yang nantinya akan digunakan sebagai *FILTER REGEX* pada *SPARQL Query*. Setelah pada proses ini, kumpulan keyword yang ditemukan, kemudian akan diolah dengan dasar kamus *Wordnet* untuk mendapatkan sinonim kata dari *keyword* tersebut. Luaran yang didapatkan dari proses ini ditunjukkan pada Gambar 6.11

Synset yang didapatkan berdasarkan Gambar 6.11 akan dianalisis dengan menggunakan *Lenveshtein Distance* untuk mendapatkan *RDF Property* mana yang cocok dari *Synset* tersebut. Hasil luaran yang didapatkan ditunjukkan pada Gambar 6.12

```
summary
sum-up
variance
Rs4792311
discrepancy
variant
variability
variableness
```

Gambar 6.11: Synset Extraction Test Case 4

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
snp:summary
snp:variant_id
```

Gambar 6.12: Hasil Property Text Similarity Question Test Case 4

Candidate Property yang didapatkan kemudian akan diteruskan pada *SPARQL Template* untuk kemudian akan dieksekusi menjadi *SPARQL Query*, di mana code *SPARQL Query* ditunjukkan pada *code* 6.7

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
SELECT * WHERE
{
```

```
?s  rdf:type          snp: ;
    snp:summary       ?p ;
    snp:variant_id    ?o
    FILTER regex(?o, "Rs4792311")
    }
    LIMIT    25
```

Kode 6.7: SPARQL Query Question Test Case 4

Hasil dari eksekusi *SPARQL Query* di atas akan menjadi JSON seperti pada *code* 6.8

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/20
            " } ,
        "p": { "type": "literal" , "value": "Increased risk of prostate
            cancer" } ,
        "o": { "type": "literal" , "value": "Rs4792311" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/29
            " } ,
        "p": { "type": "literal" , "value": "Increased risk of prostate
            cancer" } ,
        "o": { "type": "literal" , "value": "Rs4792311" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/46
            " } ,
        "p": { "type": "literal" , "value": "Normal risk of prostate
            cancer" } ,
        "o": { "type": "literal" , "value": "Rs4792311" }
      }
    ]
  }
}
```

Kode 6.8: Hasil JSON SPARQL Query Test Case 4

Question Test Case 5:
Question : What are the causes of disease Prostate Cancer?

Hasil dari POS Tagging Process pada Test Case 5 adalah sebagai berikut:

```
What do you want to ask about?
What are the causes of disease Prostate Cancer?
The following keyword have been found : causes
The following keyword have been found : disease
The following keyword have been found : Prostate
The following keyword have been found : Cancer
```

Gambar 6.13: POS Tagging Test Case 5

Dari gambar 6.13 didapatkan keyword yang ditemukan, yaitu *causes, disease, Prostate, Cancer*, semua keyword ini kemudian akan dianalisis secara semantik pada proses *Synset Extraction* untuk menemukan padanan kata atau sinonimnya. Hasil dari proses ini ditunjukkan pada gambar 6.14

```
cause
campaign
crusade
drive
movement
effort
causes
disease
Prostate
Cancer
prostate gland
prostate
cancer
malignant neoplastic disease
Cancer the Crab
Crab
```

Gambar 6.14: Synset Extraction Test Case 5

Dari kumpulan Synset ini, kemudian akan dianalisa *text similarity* nya dengan menggunakan algoritma *Levenshtein Distance* yang menghasilkan *candidate property* seperti pada gambar 6.15

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
che:causes
che:disease
gama:disease
```

Gambar 6.15: Hasil Property Text Similarity Question Test Case 5

Dari hasil di atas, bisa diketahui bahwa, terdapat property yang berbeda prefix datasetnya. Hal ini bisa terjadi dikarenakan pada proses *text similarity* keduanya sama-sama menggunakan :disease sebagai prefixnya, gama:disease dan che:disease. Namun dikarenakan pada urutan pertama adalah adalah che:causes, yang artinya jika konteks pertanyaan lebih merujuk pada menanyakan mengenai causes maka yang akan dimasukkan pada SPARQL Template adalah che:causes dan che:disease.

```
PREFIX  haz: <http://localhost:3030/hazmap/>
PREFIX  che: <http://localhost:3030/che/>
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw: <http://localhost:3030/biw/>
PREFIX  ctd: <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp: <http://localhost:3030/snpedia/>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
 SELECT  * WHERE  {
 ?s  rdf:type  che: ;
     che:causes  ?p ;
     che:disease  ?o
     FILTER regex(?o, "Cancer")
     }
     LIMIT  25
```

Kode 6.9: SPARQL Query Question Test Case 5

SPARQL Query pada code 6.9 akan dieksekusi dengan menggunakan input dari proses sebelumnya. SPARQL Query ini menghasilkan JSON seperti pada code 6.10


```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/2" } ,
        "p": { "type": "literal" , "value": "aromatic␣amines" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/3" } ,
        "p": { "type": "literal" , "value": "methyl␣bromide" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/4" } ,
        "p": { "type": "literal" , "value": "organochlorine␣pesticides" }
          ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/5" } ,
        "p": { "type": "literal" , "value": "PAHs" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/6" } ,
        "p": { "type": "literal" , "value": "pesticides" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/7" } ,
        "p": { "type": "literal" , "value": "solvents␣" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/8" } ,
        "p": { "type": "literal" , "value": "acrylonitrile" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/9" } ,
        "p": { "type": "literal" , "value": "androgens" } ,
        "o": { "type": "literal" , "value": "Prostate␣Cancer" }
```

```
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/10" }
      ,
  "p": { "type": "literal" , "value": "atrazine" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/11" }
      ,
  "p": { "type": "literal" , "value": "bisphenol␣A" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/12" }
      ,
  "p": { "type": "literal" , "value": "cadmium␣" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/13" }
      ,
  "p": { "type": "literal" , "value": "chlorophenols" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/14" }
      ,
  "p": { "type": "literal" , "value": "chromium␣" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/15" }
      ,
  "p": { "type": "literal" , "value": "DDT/DDE" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/16" }
      ,
  "p": { "type": "literal" , "value": "dibromochloropropane␣(DBCP)"
       } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/17" }
      ,
  "p": { "type": "literal" , "value": "dichlorvos␣" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/18" }
      ,
  "p": { "type": "literal" , "value": "diesel␣exhaust␣" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/19" }
      ,
  "p": { "type": "literal" , "value": "estrogens␣/␣DES" } ,
  "o": { "type": "literal" , "value": "Prostate␣Cancer" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/20" }
```

```
            ,
            "p": { "type": "literal" , "value": "methylene␣chloride␣" } ,
            "o": { "type": "literal" , "value": "Prostate␣Cancer" }
          } ,
          {
            "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/21" }
            ,
            "p": { "type": "literal" , "value": "nickel" } ,
            "o": { "type": "literal" , "value": "Prostate␣Cancer" }
          } ,
          {
            "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/22" }
            ,
            "p": { "type": "literal" , "value": "PCBs␣(polychlorinated␣
              biphenyls),␣not␣otherwise␣specified␣" } ,
            "o": { "type": "literal" , "value": "Prostate␣Cancer" }
          } ,
          {
            "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/23" }
            ,
            "p": { "type": "literal" , "value": "pesticides" } ,
            "o": { "type": "literal" , "value": "Prostate␣Cancer" }
          } ,
          {
            "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/24" }
            ,
            "p": { "type": "literal" , "value": "phenoxyacetic␣herbicides" }
            ,
            "o": { "type": "literal" , "value": "Prostate␣Cancer" }
          } ,
          {
            "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/25" }
            ,
            "p": { "type": "literal" , "value": "PhIP␣(2-amino-1-methyl-6-
              phenylimidazol(4,5-b)pyridine)␣" } ,
            "o": { "type": "literal" , "value": "Prostate␣Cancer" }
          }
        ]
      }
    }
```


Kode 6.10: Hasil JSON SPARQL Query Test Case 5

Question Test Case 6:
Question : What is the symbol of chemical pyrimidine?

Hasil dari POS Tagging Process pada Test Case 6 adalah sebagai berikut yang ditunjukkan gambar 6.16:

Dari gambar kumpulan *keyword* tersebut kemudian akan diproses menggunakan kamus *Wordnet* untuk mendapatkan padanan kata atau sinonim kata yang ditunjukkan pada Gambar 6.17

```
What do you want to ask about?
What is the evidence of the causes of methyl bromide?
The following keyword have been found : evidence
The following keyword have been found : causes
The following keyword have been found : methyl
The following keyword have been found : bromide
```

Gambar 6.16: POS Tagging Test Case 6

Dengan hasil *Synset* di atas, maka yang dilakukan kemudian adalah menemukan property RDF yang memiliki tingkat kecocokan tertinggi dan mengekstraknya. Proses ini memanfaatkan algoritma *Levenshtein Distance*. Hasil ditunjukkan pada Gambar6.18

Kemudian dua *candidate property* tersebut akan menjadi input pada parameter di *SPARQL Template*. *SPARQL Template* kemudian akan dieksekusi menjadi SPARQL Query, seperti ditunjukkan pada *code* 6.11

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
SELECT * WHERE
{
?s rdf:type che: ;
   che:evidence ?p ;
   che:causes ?o
   FILTER regex(?o, "bromide")
   }
   LIMIT 25
```

Kode 6.11: SPARQL Query Question Test Case 6

Hasil dari eksekusi query pada *code* 6.11 adalah JSON pada *code* 6.12

```
evidence
grounds
causes
methyl
bromide
cause
campaign
crusade
drive
movement
effort
methyl group
methyl radical
```

Gambar 6.17: Synset Extraction Test Case 6


```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/3" } ,
        "p": { "type": "literal" , "value": "good" } ,
        "o": { "type": "literal" , "value": "methyl_bromide" }
      }
    ]
  }
}
```


Kode 6.12: Hasil JSON SPARQL Query Test Case 6

Question Test Case 7:
Question : What is the symbol of chemical pyrimidine?

Hasil dari POS Tagging Process pada Test Case 7 adalah sebagai berikut yang ditunjukkan gambar 6.19:

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
che:evidence
che:causes
```

Gambar 6.18: Hasil Property Text Similarity Question Test Case 6

```
What do you want to ask about?
What is the symbol of chemical pyrimidine?
The following keyword have been found : symbol
The following keyword have been found : chemical
The following keyword have been found : pyrimidine
```

Gambar 6.19: POS Tagging Test Case 7

Dari hasil POS Tagging, keyword yang didapatkan adalah *symbol, chemical, pyrimidine* . Keyword ini kemudian akan dianalisa secara semantik untuk mendapatkan padanan kata dari kumpulan keyword tersebut, padanan kata atau Synset. Synset yang dihasilkan dari keyword tersebut, ditunjukkan pada gambar 6.20

```
symbol
chemical
pyrimidine
chemical substance
```

Gambar 6.20: Synset Extraction Test Case 7

Synset yang didapatkan dari Question Test Case 7 adalah: symbol, chemical, pyrimidine dan chemical substance. Kumpulan Synset ini kemudian akan dicocokan menggunakan Levenshtein Distance, mana yang memiliki kemiripan yang paling dekat, di mana hasilnya

ditunjukkan pada gambar 6.21

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
ctd:symbol
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.21: Hasil Property Text Similarity Question Test Case 7

Dapat dilihat pada gambar 6.21 terdapat beberapa property dari 2 dataset, yaitu ctd dan hazmap. Hal ini terjadi karena keyword chemical memiliki kecocokan yang tinggi dengan property ctd:chemicalname dan haz:chemicalname, sehingga keduanya terambil. Namun karena pada urutan pertama termasuk pada dataset ctd yaitu property ctd:symbol, artinya pertanyaan lebih condong menanyakan objek pada dataset ctd, sehingga property selain ctd akan dibuang dari *candidate property*. Sehingga hasil dari proses ini kemudian akan dilanjutkan dengan pembentukan dan eksekusi SPARQL Query seperti ditunjukkan pada code 6.13 dan luaran yang dihasilkan pada code 6.14

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
SELECT * WHERE {
?s rdf:type ctd: ;
   ctd:symbol ?p ;
   ctd:chemicalname ?o
   FILTER regex(?o, "pyrimidine")
```

```
        }
        LIMIT 25
```

Kode 6.13: SPARQL Query Question Test Case 7


```
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/492" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/493" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/494" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/495" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/496" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/497" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/498" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/499" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/500" } ,
```

```
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/501" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/502" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/503" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/504" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/505" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/506" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/507" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/508" } ,
      "p": { "type": "literal" , "value": "AKT1" } ,
      "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
           methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
           pyrimidine" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/509" } ,
      "p": { "type": "literal" , "value": "AR" } ,
```

```
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/510" } ,
        "p": { "type": "literal" , "value": "ARAF" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/511" } ,
        "p": { "type": "literal" , "value": "BAD" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/512" } ,
        "p": { "type": "literal" , "value": "BAD" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/513" } ,
        "p": { "type": "literal" , "value": "BAD" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/514" } ,
        "p": { "type": "literal" , "value": "BAD" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/515" } ,
        "p": { "type": "literal" , "value": "BCL2" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/516" } ,
        "p": { "type": "literal" , "value": "BCL2" } ,
        "o": { "type": "literal" , "value": "2-(1H-indazol-4-yl)-6-(4-
              methanesulfonylpiperazin-1-ylmethyl)-4-morpholin-4-ylthieno(3,2-d)
              pyrimidine" }
      }
    ]
  }
}
```


Kode 6.14: Hasil JSON SPARQL Query Test Case 7

Dari hasil JSON code 6.14 didapatkan masih terdapat nilai yang

tercetak duplikat, hal ini dikarenakan, nilai tersebut muncul pada triple lain namun berbeda property di dataset tersebut.

Question Test Case 8:
Question : What are the interactions of chemical tetrachlorobiphenyl?

Hasil dari POS Tagging Process pada Test Case 8 adalah sebagai berikut yang ditunjukkan gambar 6.22:

```
What do you want to ask about?
What are the interactions of chemical tetrachlorobiphenyl?
The following keyword have been found : interactions
The following keyword have been found : chemical
The following keyword have been found : tetrachlorobiphenyl
```

Gambar 6.22: POS Tagging Test Case 8

Keyword yang sudah didapatkan dari *Question Test Case* 8 ditunjukkan pada Gambar 6.22. *Keyword* ini kemudian akan diperluas lagi makna nya dengan menggunakan *Wordnet*. Wordnet akan mengekstrak sinonim kata dari *keyword* tersebut. Hasil dari proses ini ditunjukkan pada Gambar 6.23

```
interaction
interactions
chemical
tetrachlorobiphenyl
chemical substance
```

Gambar 6.23: Synset Extraction Test Case 8

Sinonim yang didapatk sesuai pada Gambar 6.23 kemudian akan dicari kecocokannya dengan *list Property RDF*, sehingga hasil yang

didapatkan adalah *candidate property* sesuai pada Gambar 6.24

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
ctd:interaction
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.24: Hasil Property Text Similarity Question Test Case 8

Candidate Property tersebut kemudian akan menjadi input pada *SPARQL Template* sehingga, *template* ini bisa dieksekusi menjadi query seperti pada *code* 6.15

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{ ?s  rdf:type          ctd: ;
      ctd:interaction   ?p ;
      ctd:chemicalname  ?o
      FILTER regex(?o, "tetrachlorobiphenyl")
      }
      LIMIT   25
```

Kode 6.15: SPARQL Query Question Test Case 8

Query yang dieksekusi kemudian akan menghasilkan nilai JSON seperti pada *code* 6.16

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
```

```
} ,
"results": {
  "bindings": [
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1132" } ,
      "p": { "type": "literal" , "value": "2,2',4,5'-tetrachlorobiphenyl␣
           inhibits␣the␣reaction␣[Metribolone␣results␣in␣increased␣activity␣of
           ␣AR␣protein]" } ,
      "o": { "type": "literal" , "value": "2,2',4,5'-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1685" } ,
      "p": { "type": "literal" , "value": "2,3',4,4'-tetrachlorobiphenyl␣
           inhibits␣the␣reaction␣[Metribolone␣results␣in␣increased␣activity␣of
           ␣AR␣protein]" } ,
      "o": { "type": "literal" , "value": "2,3',4,4'-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1687" } ,
      "p": { "type": "literal" , "value": "2,3,4',6-tetrachlorobiphenyl␣
           inhibits␣the␣reaction␣[Metribolone␣results␣in␣increased␣activity␣of
           ␣AR␣protein]" } ,
      "o": { "type": "literal" , "value": "2,3,4',6-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1691" } ,
      "p": { "type": "literal" , "value": "2,3,5,6-tetrachlorobiphenyl␣inhibits
           ␣the␣reaction␣[Metribolone␣results␣in␣increased␣activity␣of␣AR␣
           protein]" } ,
      "o": { "type": "literal" , "value": "2,3,5,6-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1752" } ,
      "p": { "type": "literal" , "value": "[aroclor␣1260␣co-treated␣with␣
           Chlorodiphenyl␣(54␣percent␣Chlorine)␣co-treated␣with␣2,4,4'-
           trichlorobiphenyl␣co-treated␣with␣2,4,2',4'-tetrachlorobiphenyl␣co-
           treated␣with␣3,4,5,3',4'-pentachlorobiphenyl␣co-treated␣with␣
           3,4,3',4'-tetrachlorobiphenyl]␣inhibits␣the␣reaction␣[
           Dihydrotestosterone␣results␣in␣increased␣expression␣of␣AR␣mRNA]" }
           ,
      "o": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1753" } ,
      "p": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl␣
           results␣in␣increased␣phosphorylation␣of␣MAPK1␣protein" } ,
      "o": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1754" } ,
      "p": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl␣
           results␣in␣increased␣phosphorylation␣of␣MAPK3␣protein" } ,
      "o": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1755" } ,
      "p": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl␣
           results␣in␣increased␣phosphorylation␣of␣MDM2␣protein" } ,
      "o": { "type": "literal" , "value": "2,4,2',4'-tetrachlorobiphenyl" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1759" } ,
      "p": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl␣
           inhibits␣the␣reaction␣[Metribolone␣results␣in␣increased␣activity␣of
```

```
␣AR␣protein]" } ,
  "o": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1760" } ,
  "p": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl␣results
       ␣in␣increased␣phosphorylation␣of␣MAPK1␣protein" } ,
  "o": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1761" } ,
  "p": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl␣results
       ␣in␣increased␣phosphorylation␣of␣MAPK3␣protein" } ,
  "o": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/1762" } ,
  "p": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl␣results
       ␣in␣increased␣phosphorylation␣of␣MDM2␣protein" } ,
  "o": { "type": "literal" , "value": "2,4,4',5-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2151" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       inhibits␣the␣reaction␣[Dihydrotestosterone␣results␣in␣increased␣
       activity␣of␣AR␣protein]" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2152" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       results␣in␣increased␣expression␣of␣CDK2␣mRNA" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2153" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       results␣in␣increased␣expression␣of␣GSTP1␣mRNA" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2154" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       results␣in␣increased␣expression␣of␣HSP90AB1␣mRNA" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2155" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       affects␣the␣expression␣of␣IGF1R␣mRNA" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2156" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       results␣in␣decreased␣phosphorylation␣of␣MAPK1␣protein" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2157" } ,
  "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
       results␣in␣decreased␣phosphorylation␣of␣MAPK3␣protein" } ,
  "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
} ,
```

```
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2158" } ,
        "p": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl␣
             results␣in␣increased␣phosphorylation␣of␣MDM2␣protein" } ,
        "o": { "type": "literal" , "value": "2,5,2',5'-tetrachlorobiphenyl" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2515" } ,
        "p": { "type": "literal" , "value": "[aroclor␣1260␣co-treated␣with␣
             Chlorodiphenyl␣(54␣percent␣Chlorine)␣co-treated␣with␣2,4,4'-
             trichlorobiphenyl␣co-treated␣with␣2,4,2',4'-tetrachlorobiphenyl␣co-
             treated␣with␣3,4,5,3',4'-pentachlorobiphenyl␣co-treated␣with␣
             3,4,3',4'-tetrachlorobiphenyl]␣inhibits␣the␣reaction␣[
             Dihydrotestosterone␣results␣in␣increased␣expression␣of␣AR␣mRNA]" }
              ,
        "o": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2516" } ,
        "p": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl␣
             inhibits␣the␣reaction␣[INS␣protein␣results␣in␣decreased␣abundance␣
             of␣Blood␣Glucose]" } ,
        "o": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2517" } ,
        "p": { "type": "literal" , "value": "AHR␣protein␣promotes␣the␣reaction␣
             [3,4,3',4'-tetrachlorobiphenyl␣inhibits␣the␣reaction␣[INS␣protein␣
             results␣in␣decreased␣abundance␣of␣Blood␣Glucose]]" } ,
        "o": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2518" } ,
        "p": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl␣
             promotes␣the␣reaction␣[[AHR␣protein␣binds␣to␣ARNT␣protein]␣which␣
             binds␣to␣RAF1␣promoter]" } ,
        "o": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/ctd/2519" } ,
        "p": { "type": "literal" , "value": "resveratrol␣inhibits␣the␣reaction␣
             [3,4,3',4'-tetrachlorobiphenyl␣promotes␣the␣reaction␣[[AHR␣protein␣
             binds␣to␣ARNT␣protein]␣which␣binds␣to␣RAF1␣promoter]]" } ,
        "o": { "type": "literal" , "value": "3,4,3',4'-tetrachlorobiphenyl" }
      }
    ]
  }
}
```


Kode 6.16: Hasil JSON SPARQL Query Test Case 8

Question Test Case 9:
Question : What is the synonyms of chemical chlorophenoxybutyric?

Hasil dari POS Tagging Process pada Test Case 9 adalah sebagai

berikut yang ditunjukkan gambar 6.25:

```
What do you want to ask about?
What is the synonyms of chemical chlorophenoxybutyric?
The following keyword have been found : synonyms
The following keyword have been found : chemical
The following keyword have been found : chlorophenoxybutyric
```

Gambar 6.25: POS Tagging Test Case 9

Dari keyword pada Gabar 6.25, akan diperluas lagi dengan menemukan atau mengekstrak sinonim katanya. Hasil atau luaran dari proses ini ditunjukkan pada Gambar 6.26

```
synonym
equivalent word
synonyms
chemical
chlorophenoxybutyric
chemical substance
```

Gambar 6.26: Synset Extraction Test Case 9

Kumpulan dari sinonim kata atau disebut dengan *synset* ini kemudian akan diukur kemiripannya dengan string pada *list RDF Property* dengan menggunakan *Levenshtein Distance*. Property dengan nilai kecocokan tertinggi akan diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.27

Dari *candidate property* yang didapatkan, terlihat jika ada property dari *class* dataset lain yang terekstrak. Namun karena pada urutan pertama merupakan RDF Property dari *hazmap*, sehingga menerangkan jika pertanyaan tersebut memiliki konteks pada dataset *hazmap*, maka candidate property selain *hazmap* akan dibuang dari *candidate property*. Sehingga *code* query yang dihasilkan ditun-

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
haz:synonims
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.27: Hasil Property Text Similarity Question Test Case 9

jukkan pada *code* 6.17

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{ ?s  rdf:type          ctd: ;
      ctd:interaction   ?p ;
      ctd:chemicalname  ?o
      FILTER regex(?o, "tetrachlorobiphenyl")
      }
      LIMIT   25
```

Kode 6.17: SPARQL Query Question Test Case 9

Dari hasil query *code* 6.17 dihasilkan nilai JSON pada *code* 6.18

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/haz/2" } ,
        "p": { "type": "literal" , "value": "MCPB; (4-Chloro-o-tolyloxy)
              butyric acid; 2,4-MCPB; 2M 4KhM; 4-((4-Chloro-o-tolyl)oxy)
              butyric acid; 4-(2-Methyl-4-chlorophenoxy)butyric acid;
```

```
                4-(2-Methyl-4-chlorphenoxy)-buttersaeure␣[German];␣4-(4-
                Chlor-2-methylphenoxy)-buttersaeure␣[German];␣4-(4-Chloro
                -2-methylphenoxy)butanoic␣acid;␣4-(4-Chloro-2-methylphenoxy
                )butyric␣acid;␣4MCPB;␣Bexone;␣Kyselina␣4-(4-chlor-2-
                methylfenoxy)maselna␣[Czech];␣Legumex;␣MCP-butyric;␣PDQ;␣
                Thitrol;␣Trifolex;␣Trotox;␣U46␣MCPB;␣gamma-(4-Chloro-2-
                methylphenoxy)butyric␣acid;␣gamma-MCPB;␣[ChemIDplus]" } ,
          "o": { "type": "literal" , "value": "2-Methyl-4-
                chlorophenoxybutyric␣acid" }
        }
      ]
    }
  }
```


Kode 6.18: Hasil JSON SPARQL Query Test Case 9

Question Test Case 10:
Question : What is the formula of chemical chlorophenoxybutyric?

Hasil dari POS Tagging Process pada Test Case 10 adalah sebagai berikut yang ditunjukkan Gambar 6.28:

```
What do you want to ask about?
What is the formula of chemical chlorophenoxybutyric?
The following keyword have been found : formula
The following keyword have been found : chemical
The following keyword have been found : chlorophenoxybutyric
```

Gambar 6.28: POS Tagging Test Case 10

Keyword yang ditemukan berupa *formula, chemical* dan *chlorophenoxybutyric*. Keyword tersebut kemudian akan dicari sinonim katanya pada kamus *Wordnet*. Hasil dari proses ini ditunjukkan pada Gambar 6.29

Synset yang sudah didapatkan kemudian akan ditemukan nilai kemiripannya dengan *RDF Property*. *Property* yang memiliki nilai kecocokan tertinggi akan diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.30

```
formula
expression
chemical formula
rule
chemical
chlorophenoxybutyric
chemical substance
```

Gambar 6.29: Synset Extraction Test Case 10

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
haz:formula
gama:region
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.30: Hasil Property Text Similarity Question Test Case 10

Sama dengan hasil pengujian sebelumnya, *candidate property* yang didapatkan memiliki juga property lebih dari 1 dataset. Namun karena pada urutan pertama *candidate property* merupakan *property* dari dataset *hazmap*, maka mengindikasikan jika pertanyaan lebih condong pada dataset *hazmap*. Oleh karena itu, property yang bukan dari *class hazmap* akan dibuang dari *candidate property*. Kemudian nilai ini kemudian akan dimasukkan pada *SPARQL Template*, sehingga *code* nya ditunjukkan pada *code* 6.19

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw: <http://localhost:3030/biw/>
PREFIX  ctd: <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp: <http://localhost:3030/snpedia/>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{
?s  rdf:type            haz: ;
    haz:formula         ?p ;
    haz:chemicalname    ?o
    FILTER regex(?o, "chlorophenoxybutyric")
    }
    LIMIT   25
```

Kode 6.19: SPARQL Query Question Test Case 10

Hasil dari eksekusi *query* ini ditunjukkan pada *code* 6.20

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/haz/2" } ,
        "p": { "type": "literal" , "value": "C11-H13-Cl-O3" } ,
        "o": { "type": "literal" , "value": "2-Methyl-4-
              chlorophenoxybutyric_acid" }
      }
    ]
  }
}
```

Kode 6.20: Hasil JSON SPARQL Query Test Case 10

Question Test Case 11:
Question : What is the category of chemical Metaldehyde?

Hasil dari POS Tagging Process pada Test Case 11 adalah sebagai berikut yang ditunjukkan Gambar 6.31:

Keyword kemudian akan diperluas makna semantiknya dengan menggunakan *Wordnet*. Hasil dari proses ini ditunjukkan pada Gambar 6.32

```
What do you want to ask about?
What is the category of chemical Metaldehyde?
The following keyword have been found : category
The following keyword have been found : chemical
The following keyword have been found : Metaldehyde
```

Gambar 6.31: POS Tagging Test Case 11

```
class
category
family
chemical
Metaldehyde
chemical substance
```

Gambar 6.32: Synset Extraction Test Case 11

Synset yang sudah didapatkan kemudian akan ditemukan nilai kemiripannya dengan *RDF Property*. *Property* yang memiliki nilai kecocokan tertinggi akan diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.33

Candidate Property yang didapatkan terdapat *property* yang berasal dari dataset lain. Dalam hal ini *candidate property* memiliki lebih dari 1 jenis property. Karena pada *candidate property* sendiri pada urutan pertama merupakan property dari dataset *hazmap*, maka mengindikasikan bahwa pertanyaan condong pada dataset *hazmap*. Nilai dari *candidate property* ini kemudian akan menjadi input untuk *SPARQL Template*. *Template* yang sudah memiliki nilai akan dieksekusi menjadi query. Hasil dari proses ini ditunjukkan pada *code* 6.21

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
```

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
haz:category
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.33: Hasil Property Text Similarity Question Test Case 11

```
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw: <http://localhost:3030/biw/>
PREFIX  ctd: <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp: <http://localhost:3030/snpedia/>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{ ?s  rdf:type            haz: ;
      haz:category        ?p ;
      haz:chemicalname    ?o
      FILTER regex(?o, "Metaldehyde")
      }
      LIMIT   25
```

Kode 6.21: SPARQL Query Question Test Case 11

Setelah query dieksekusi akan menghasilkan JSON seperti pada *code* 6.22

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/haz/2" } ,
        "p": { "type": "literal" , "value": "C11-H13-Cl-O3" } ,
        "o": { "type": "literal" , "value": "2-Methyl-4-
              chlorophenoxybutyric acid" }
      }
    ]
  }
```

```
}
```

Kode 6.22: Hasil JSON SPARQL Query Test Case 11

Question Test Case 12:
Question : What is the source of chemical Metaldehyde?

Hasil dari POS Tagging Process pada Test Case 12 adalah sebagai berikut yang ditunjukkan Gambar 6.34:

```
What do you want to ask about?
What is the source of chemical Metaldehyde?
The following keyword have been found : source
The following keyword have been found : chemical
The following keyword have been found : Metaldehyde
```

Gambar 6.34: POS Tagging Test Case 12

Keyword kemudian akan diperluas makna semantiknya dengan menggunakan *Wordnet*. Hasil dari proses ini ditunjukkan pada Gambar 6.35

Setelah didapatkan *synset* nya, kumpulan kata pada *synset* akan dicocokan dengan list *RDF Property*. Nilai kecocokan tertinggi akan diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.36

Candidate Property yang didapatkan mengandung 3 property dataset, yaitu *class* hazmap, gama, dan ctd. Hal ini terjadi karena property dataset tersebut memiliki kesamaan, yaitu chemicalname dan chemical id, baik dari dataset hazmap dan dataset ctd. Sehingga property tersebut terambil. Karena pada urutan pertama di *candidate* property adalah property dataset *haz* yaitu haz:source, sehingga mengindikasikan jika pertayaan tersebut mengacu pada konteks dataset hazmap, sehingga property selain yang berasal dari data-

```
beginning
origin
root
rootage
source
generator
author
chemical
Metaldehyde
chemical substance
```

Gambar 6.35: Synset Extraction Test Case 12

set hazmap dibuang dari candidate property. Setelah itu nilai dari *candidate property* akan menjadi input dari *SPARQL Template* dan dieksekusi, seperti ditunjukkan pada *code* 6.23

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
SELECT * WHERE
{ ?s rdf:type haz: ;
    haz:source ?p ;
    haz:chemicalname ?o
    FILTER regex(?o, "Metaldehyde")
    }
    LIMIT 25
```

Kode 6.23: SPARQL Query Question Test Case 12

Hasil dari eksekusi query *code* 6.23 ditunjukkan pada nilai JSON pada *code* 6.24

```
{
```

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
haz:source
gama:or
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.36: Hasil Property Text Similarity Question Test Case 12


```
"head": {
  "vars": [ "s" , "p" , "o" ]
} ,
"results": {
  "bindings": [
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/haz/3" } ,
      "p": { "type": "literal" , "value": "Used␣for␣control␣of␣snails␣
           in␣seed␣crops,␣turf,␣ornamentals,␣berries,␣citrus,␣
           vegetables,␣and␣other␣crops;␣[Reference␣#1]␣Used␣for␣
           synthesis␣in␣the␣chemical␣industry,␣as␣odor␣agent,␣and␣as␣
           pesticide;␣[IUCLID]␣Used␣as␣molluscicide,␣solid␣fuel␣for␣
           small␣heaters␣(alcohol␣replacement),␣and␣fire␣starter;␣[
           Reference␣#2]" } ,
      "o": { "type": "literal" , "value": "Metaldehyde" }
    }
  ]
}
}
```


Kode 6.24: Hasil JSON SPARQL Query Test Case 12

Question Test Case 13:
Question : What is the cas number of chemical propylphthalate?

Hasil dari POS Tagging Process pada Test Case 13 adalah sebagai berikut yang ditunjukkan Gambar 6.37:

Keyword kemudian akan diperluas makna semantiknya dengan meng-

```
What do you want to ask about?
What is the cas number of chemical propylphthalate?
The following keyword have been found : cas
The following keyword have been found : number
The following keyword have been found : chemical
The following keyword have been found : propylphthalate
```

Gambar 6.37: POS Tagging Test Case 13

gunakan *Wordnet*. Hasil dari proses ini ditunjukkan pada Gambar 6.38

```
calcium
Ca
atomic number 20
cas
number
chemical
propylphthalate
figure
identification number
numeral
chemical substance
```

Gambar 6.38: Synset Extraction Test Case 13

Setelah didapatkan *synset* nya, kumpulan kata pada *synset* akan dicocokan dengan list *RDF Property*. Nilai kecocokan tertinggi akan diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.39

Candidate Property yang didapatkan merupakan kumpulan property dari dataset hazmap. Karena pada urutan pertama pada *candidate property* merupakan property hazmap, yaitu haz:cas, maka mengindikasikan bahwa pertanyaan mengandung konteks dari dataset ha-

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
haz:cas
haz:chemicalname
ctd:chemicalname
ctd:chemicalid
```

Gambar 6.39: Hasil Property Text Similarity Question Test Case 13

zmap. Kemudian nilai tersebut akan menjadi input pada *SPARQL Template*. Template yang sudah mendapat nilai, akan dieksekusi menjadi *SPARQL Query*, seperti ditunjukkan pada *code* 6.25

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{ ?s  rdf:type          haz: ;
      haz:cas           ?p ;
      haz:chemicalname  ?o
      FILTER regex(?o, "propylphthalate")
      }
      LIMIT   25
```

Kode 6.25: SPARQL Query Question Test Case 13

Hasil dari *query* pada *code* 6.25 ditunjukkan dari nilai JSON *code* 6.26

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
```

```
            {
              "s": { "type": "uri" , "value": "http://127.0.0.1:3333/haz/3" } ,
              "p": { "type": "literal" , "value": "Used␣for␣control␣of␣snails␣
                   in␣seed␣crops,␣turf,␣ornamentals,␣berries,␣citrus,␣
                   vegetables,␣and␣other␣crops;␣[Reference␣#1]␣Used␣for␣
                   synthesis␣in␣the␣chemical␣industry,␣as␣odor␣agent,␣and␣as␣
                   pesticide;␣[IUCLID]␣Used␣as␣molluscicide,␣solid␣fuel␣for␣
                   small␣heaters␣(alcohol␣replacement),␣and␣fire␣starter;␣[
                   Reference␣#2]" } ,
              "o": { "type": "literal" , "value": "Metaldehyde" }
            }
          ]
        }
      }
```


Kode 6.26: Hasil JSON SPARQL Query Test Case 13

Question Test Case 14:
Question : Who is the researcher that has published journal Nat Genet?

Hasil dari POS Tagging Process pada Test Case 14 adalah sebagai berikut yang ditunjukkan Gambar 6.40:

```
What do you want to ask about?
Who is the researcher that has published journal Nat Genet?
The following keyword have been found : researcher
The following keyword have been found : journal
The following keyword have been found : Nat
The following keyword have been found : Genet
```

Gambar 6.40: POS Tagging Test Case 14

Keyword yang didapatkan pada tahap *POS Tagging* adalah : *researcher, journal, Nat, Genet*. Keyword ini kemudian akan diperluas makna semantiknya dengan menambah sinonim kata dari *keyword* tersebut dengan mencari *synset* nya pada kamus *Wordnet*. Luaran dari tahap ini ditunjukkan pada Gambar 6.41

Setelah didapatkan *synset* nya, kumpulan kata pada *synset* akan dicocokan dengan list *RDF Property*. Nilai kecocokan tertinggi akan

```
research worker
researcher
investigator
journal
Nat
Genet
diary
Edmund Charles Edouard Genet
Citizen Genet
```

Gambar 6.41: Synset Extraction Test Case 14

diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.42

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
pubmed:researcher
pubmed:journal
```

Gambar 6.42: Hasil Property Text Similarity Question Test Case 14

Candidate Property kemudian akan dijadikan input untuk *SPARQL Template*. Template yang sudah mendapatkan nilai akan dieksekusi, ditunjukkan pada *code* 6.27

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
```

```
SELECT * WHERE
{
?s  rdf:type            gama: ;
    pubmed:researcher   ?p ;
    pubmed:journal      ?o
    FILTER regex(?o, "Genet")
    }
    LIMIT   25
```

Kode 6.27: SPARQL Query Question Test Case 14

Hasil dari query pada *code*6.27, akan menghasilkan nilai JSON pada *code* 6.28


```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/287" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/237" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/276" } ,
        "p": { "type": "literal" , "value": "Thomas␣G" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/232" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/286" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/289" } ,
        "p": { "type": "literal" , "value": "Yeager␣M" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/226" } ,
        "p": { "type": "literal" , "value": "Takata␣R" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/258" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
```

```
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/238" } ,
  "p": { "type": "literal" , "value": "Eeles␣RA" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/260" } ,
  "p": { "type": "literal" , "value": "Eeles␣RA" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/284" } ,
  "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/182" } ,
  "p": { "type": "literal" , "value": "Schumacher␣FR" } ,
  "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/227" } ,
  "p": { "type": "literal" , "value": "Takata␣R" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/270" } ,
  "p": { "type": "literal" , "value": "Eeles␣RA" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/193" } ,
  "p": { "type": "literal" , "value": "Schumacher␣FR" } ,
  "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/278" } ,
  "p": { "type": "literal" , "value": "Thomas␣G" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/184" } ,
  "p": { "type": "literal" , "value": "Schumacher␣FR" } ,
  "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/288" } ,
  "p": { "type": "literal" , "value": "Yeager␣M" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/272" } ,
  "p": { "type": "literal" , "value": "Thomas␣G" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/261" } ,
  "p": { "type": "literal" , "value": "Eeles␣RA" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/274" } ,
  "p": { "type": "literal" , "value": "Thomas␣G" } ,
  "o": { "type": "literal" , "value": "Nat␣Genet" }
```

```
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/234" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/285" } ,
        "p": { "type": "literal" , "value": "Gudmundsson␣J" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/105" } ,
        "p": { "type": "literal" , "value": "Amin␣Al␣Olama␣A" } ,
        "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/262" } ,
        "p": { "type": "literal" , "value": "Eeles␣RA" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      }
    ]
  }
}
```


Kode 6.28: Hasil JSON SPARQL Query Test Case 14

Question Test Case 15:
Question : Who is the researcher that has published journal Nat Genet?

Hasil dari POS Tagging Process pada Test Case 15 adalah sebagai berikut yang ditunjukkan Gambar 6.43:

```
What do you want to ask about?
When is publication of journal Nat Genet?
The following keyword have been found : publication
The following keyword have been found : journal
The following keyword have been found : Nat
The following keyword have been found : Genet
```

Gambar 6.43: POS Tagging Test Case 15

Keyword yang didapatkan pada tahap *POS Tagging* adalah : *publication, journal, Nat, Genet*. Keyword ini kemudian akan diperluas

makna semantiknya dengan menambah sinonim kata dari *keyword* tersebut dengan mencari *synset* nya pada kamus *Wordnet*. Luaran dari tahap ini ditunjukkan pada Gambar 6.44

```
publication
publishing
journal
Nat
Genet
diary
Edmund Charles Edouard Genet
Citizen Genet
```

Gambar 6.44: Synset Extraction Test Case 15

Setelah didapatkan *synset* nya, kumpulan kata pada *synset* akan dicocokan dengan list *RDF Property*. Nilai kecocokan tertinggi akan diekstrak. Hasil dari proses ini ditunjukkan pada Gambar 6.45

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
pubmed:publication
pubmed:journal
```

Gambar 6.45: Hasil Property Text Similarity Question Test Case 15

Karena pada *candidate property* di atas tidak mengandung proeprty dari dataset lain, maka tidak perlu adanya reduksi. Nilai-nilai pada *candidate property* kemudian akan dijadikan input pada SPARQL Template. Tempate yang sudah mengandung nilai sesuai dengan *candidate property* pada Gambar 6.45 akan dieksekusi menjadi *SPARQL Query* seperti pada *code* **??**

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{
?s  rdf:type            gama: ;
    pubmed:publication  ?p ;
    pubmed:journal      ?o
    FILTER regex(?o, "Genet")
    }
    LIMIT   25
```

Kode 6.29: SPARQL Query Question Test Case 15

Hasil dari eksekusi query *code* 6.29 dapat dilihat pada *code* JSON 6.30


```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/287" } ,
        "p": { "type": "literal" , "value": "Sun␣Apr␣01␣00:00:00␣ICT␣2007" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/237" } ,
        "p": { "type": "literal" , "value": "Sun␣Sep␣20␣00:00:00␣ICT␣2009" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/276" } ,
        "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/232" } ,
        "p": { "type": "literal" , "value": "Sun␣Sep␣20␣00:00:00␣ICT␣2009" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/286" } ,
        "p": { "type": "literal" , "value": "Sun␣Apr␣01␣00:00:00␣ICT␣2007" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
```

```
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/289" } ,
      "p": { "type": "literal" , "value": "Sun␣Apr␣01␣00:00:00␣ICT␣2007" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/226" } ,
      "p": { "type": "literal" , "value": "Sun␣Aug␣01␣00:00:00␣ICT␣2010" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/258" } ,
      "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/238" } ,
      "p": { "type": "literal" , "value": "Sun␣Sep␣20␣00:00:00␣ICT␣2009" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/260" } ,
      "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/284" } ,
      "p": { "type": "literal" , "value": "Sun␣Jul␣01␣00:00:00␣ICT␣2007" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/182" } ,
      "p": { "type": "literal" , "value": "Fri␣Jul␣08␣00:00:00␣ICT␣2011" } ,
      "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/227" } ,
      "p": { "type": "literal" , "value": "Sun␣Aug␣01␣00:00:00␣ICT␣2010" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/270" } ,
      "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/193" } ,
      "p": { "type": "literal" , "value": "Fri␣Jul␣08␣00:00:00␣ICT␣2011" } ,
      "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/278" } ,
      "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/184" } ,
      "p": { "type": "literal" , "value": "Fri␣Jul␣08␣00:00:00␣ICT␣2011" } ,
      "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/288" } ,
      "p": { "type": "literal" , "value": "Sun␣Apr␣01␣00:00:00␣ICT␣2007" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
```

```
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/272" } ,
        "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/261" } ,
        "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/274" } ,
        "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/234" } ,
        "p": { "type": "literal" , "value": "Sun␣Sep␣20␣00:00:00␣ICT␣2009" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/285" } ,
        "p": { "type": "literal" , "value": "Sun␣Jul␣01␣00:00:00␣ICT␣2007" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/105" } ,
        "p": { "type": "literal" , "value": "Fri␣Oct␣12␣00:00:00␣ICT␣2012" } ,
        "o": { "type": "literal" , "value": "Hum␣Mol␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/262" } ,
        "p": { "type": "literal" , "value": "Sun␣Feb␣10␣00:00:00␣ICT␣2008" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      }
    ]
  }
}
```


Kode 6.30: Hasil JSON SPARQL Query Test Case 15

Question Test Case 16:
Question : What is the region of researcher Eeles RA found ?

Hasil dari POS Tagging Process pada Test Case 16 adalah sebagai berikut yang ditunjukkan Gambar 6.46:

Keyword yang ditemukan dari tahap ini adalah *region, researcher, Eeles, RA*. Keyword yang terletak di paling akhir akan digunakan sebagai *FILTER REGEX* pada *SPARQL Query*. Semua keyword ini kemudian akan diperluas makna semantiknya dengan *Wordnet*.

```
What do you want to ask about?
What is the region of researcher Eeles RA found ?
The following keyword have been found : region
The following keyword have been found : researcher
The following keyword have been found : Eeles
The following keyword have been found : RA
```

Gambar 6.46: POS Tagging Test Case 16

Hasilnya dari tahapan ini ditunjukkan pada Gambar 6.47

```
region
part
neighborhood
researcher
Eeles
RA
research worker
investigator
radium
Ra
atomic number 88
```

Gambar 6.47: Synset Extraction Test Case 16

Kemudian setelah *synset* didapatkan, langkah selanutnya adalah proses *similarity measuring* untuk menemukan RDF Property mana yang cocko dengan synset tersebut. Hasil dari proses ini ditunjukkan dari Gambar 6.48

Setelah mendapatkan *candidate property* dan memastikan bahwa tidak ada lebih dari 1 dataset property yang masuk ke dalam *candidate property*, maka nilai dari *candidate property* akan menjadi

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
gama:region
pubmed:researcher
```

Gambar 6.48: Hasil Property Text Similarity Question Test Case 16

input untuk *SPARQL Template*. Hasilnya ditunjukkan dari query pada *code*6.31

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gama: <http://localhost:3030/gama/>
SELECT * WHERE
{
?s rdf:type gama: ;
   gama:region ?p ;
   pubmed:researcher ?o
   FILTER regex(?o, "RA")
   }
   LIMIT 25
```

Kode 6.31: SPARQL Query Question Test Case 16

Dari query tersebut, pada bagian *FILTER REGEX* meskipun tidak secara lengkap nama dari *researcher*, tapi tetap berjalan, karena yang digunakan adalah *REGEX* sehingga tidak perlu 1 konteks penuh dari nama *researcher*. Hasil dari query ditunjukkan dari nilai JSON pada *code* 6.32

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
```

```
"bindings": [
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/238" } ,
    "p": { "type": "literal" , "value": "17q24.3" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/260" } ,
    "p": { "type": "literal" , "value": "17q24.3" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/270" } ,
    "p": { "type": "literal" , "value": "10q11.22" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/261" } ,
    "p": { "type": "literal" , "value": "8q24.21" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/262" } ,
    "p": { "type": "literal" , "value": "17q12" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/92" } ,
    "p": { "type": "literal" , "value": "6q25.2" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/244" } ,
    "p": { "type": "literal" , "value": "4q22.3" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/256" } ,
    "p": { "type": "literal" , "value": "8p21.2" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/86" } ,
    "p": { "type": "literal" , "value": "7p15.3" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/100" } ,
    "p": { "type": "literal" , "value": "6p21.32" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/268" } ,
    "p": { "type": "literal" , "value": "11q13.3" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/98" } ,
    "p": { "type": "literal" , "value": "12q24.21" } ,
    "o": { "type": "literal" , "value": "Eeles_RA" }
  } ,
  {
    "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/251" } ,
    "p": { "type": "literal" , "value": "11p15.5" } ,
```

```
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/81" } ,
        "p": { "type": "literal" , "value": "8p21.2" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/263" } ,
        "p": { "type": "literal" , "value": "19q13.33" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/93" } ,
        "p": { "type": "literal" , "value": "Xp22.2" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/245" } ,
        "p": { "type": "literal" , "value": "4q22.3" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/87" } ,
        "p": { "type": "literal" , "value": "14q24.1" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/101" } ,
        "p": { "type": "literal" , "value": "18q23" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/269" } ,
        "p": { "type": "literal" , "value": "3p12.1" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/99" } ,
        "p": { "type": "literal" , "value": "11q22.2" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/239" } ,
        "p": { "type": "literal" , "value": "17q12" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/240" } ,
        "p": { "type": "literal" , "value": "2q31.1" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/252" } ,
        "p": { "type": "literal" , "value": "10q11.22" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/82" } ,
        "p": { "type": "literal" , "value": "2q37.3" } ,
        "o": { "type": "literal" , "value": "Eeles␣RA" }
      }
    ]
  }
```

```
}
```

Kode 6.32: Hasil JSON SPARQL Query Test Case 16

Question Test Case 17:
Question : What is the mapped gene of researcher Eeles RA found ?

Hasil dari POS Tagging Process pada Test Case 17 adalah sebagai berikut yang ditunjukkan Gambar 6.49:

```
What do you want to ask about?
What is the mapped gene of researcher Eeles RA found ?
The following keyword have been found : gene
The following keyword have been found : researcher
The following keyword have been found : Eeles
The following keyword have been found : RA
```

Gambar 6.49: POS Tagging Test Case 17

Keyword yang sudah ditemukan kemudian akan diproses menggunakan kamus *Wordnet* untuk mendapatkan kata sinonim dari keyword yang sudah ditemukan pada tahap POS Tagging. Namun dapat kita amati jika, kata mapped gene yang terekstrak hanya pada kata gene nya saja. Hal ini akan memberi pengaruh pada tahap selanjutnya.

Hasil Synset ditunjukkan pada Gambar6.50

Synset ini kemudian akan diukur tingkat kemiripannya dengna *list Property RDF* dan akan diekstrak *property* yang memiliki nilai kecocokan yang tinggi. Hasil dari proses ini ditunjukkan pada Gambar 6.51

```
gene
cistron
factor
researcher
Eeles
RA
research worker
investigator
radium
Ra
atomic number 88
```

Gambar 6.50: Synset Extraction Test Case 17

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
pubmed:researcher
```

Gambar 6.51: Hasil Property Text Similarity Question Test Case 17

Candidate Property yang didapatkan hanya pubmed:researcher saja, padahal seharusnya karena menanyakan mengenai mapped gene, maka seharusnya property gama:mapped, harus ikut terkstrak agar menjawab pertanyaan. Hal ini akan memberi pengaruh pada hasil query. SPARQL Query ditunjukkan pada *code* 6.33

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX biw: <http://localhost:3030/biw/>
PREFIX ctd: <http://localhost:3030/ctd/>
PREFIX pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX snp: <http://localhost:3030/snpedia/>
```

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{
?s  rdf:type            gama: ;
    pubmed:researcher   ?researcher
    }
    LIMIT   25
```

Kode 6.33: SPARQL Query Question Test Case 17

Query di atas tidak menjawab pertanyaan yang diajukan, sehingga hal ini akan membuat nilai jawaban tidak tepat. Hasil dari query ditunjukkan pada *code* JSON 6.34


```
{
  "head": {
    "vars": [ "s" , "researcher" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/20" } ,
        "researcher": { "type": "literal" , "value": "Berndt␣SI" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/287" } ,
        "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/237" } ,
        "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/113" } ,
        "researcher": { "type": "literal" , "value": "Cheng␣I" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/14" } ,
        "researcher": { "type": "literal" , "value": "Berndt␣SI" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/276" } ,
        "researcher": { "type": "literal" , "value": "Thomas␣G" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/232" } ,
        "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/286" } ,
        "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/289" } ,
        "researcher": { "type": "literal" , "value": "Yeager␣M" }
      } ,
      {
```

```
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/66" } ,
      "researcher": { "type": "literal" , "value": "Knipe␣DW" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/226" } ,
      "researcher": { "type": "literal" , "value": "Takata␣R" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/258" } ,
      "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/238" } ,
      "researcher": { "type": "literal" , "value": "Eeles␣RA" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/260" } ,
      "researcher": { "type": "literal" , "value": "Eeles␣RA" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/284" } ,
      "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/182" } ,
      "researcher": { "type": "literal" , "value": "Schumacher␣FR" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/68" } ,
      "researcher": { "type": "literal" , "value": "Knipe␣DW" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/227" } ,
      "researcher": { "type": "literal" , "value": "Takata␣R" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/270" } ,
      "researcher": { "type": "literal" , "value": "Eeles␣RA" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/193" } ,
      "researcher": { "type": "literal" , "value": "Schumacher␣FR" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/278" } ,
      "researcher": { "type": "literal" , "value": "Thomas␣G" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/78" } ,
      "researcher": { "type": "literal" , "value": "Lange␣EM" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/19" } ,
      "researcher": { "type": "literal" , "value": "Berndt␣SI" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/184" } ,
      "researcher": { "type": "literal" , "value": "Schumacher␣FR" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/288" } ,
      "researcher": { "type": "literal" , "value": "Yeager␣M" }
    }
  ]
```

```
  }
}
```

Kode 6.34: Hasil JSON SPARQL Query Test Case 17

Hasil dari query di atas tidak menjawab pertanyaan. Hal ini terjadi karena kesalahan awal yang terjadi pada POS Tagging karena tidak dapat mengekstrak kata dengan tepat.

Question Test Case 18:
Question : What is the reported gene of researcher Eeles RA found?

Hasil dari POS Tagging Process pada Test Case 18 adalah sebagai berikut yang ditunjukkan Gambar 6.52:

```
What do you want to ask about?
What is the reported gene of researcher Eeles RA found?
The following keyword have been found : gene
The following keyword have been found : researcher
The following keyword have been found : Eeles
The following keyword have been found : RA
```

Gambar 6.52: POS Tagging Test Case 18

Keyword yang sudah ditemukan kemudian akan diproses menggunakan kamus *Wordnet* untuk mendapatkan kata sinonim dari keyword yang sudah ditemukan pada tahap POS Tagging. Namun dapat kita amati jika, kata mapped gene yang terekstrak hanya pada kata gene nya saja. Hal ini akan memberi pengaruh pada tahap selanjutnya. Hasil yang dihasilkan dari pertanyaan ini sama dengan *Test Case 17*

Hasil Synset ditunjukkan pada Gambar6.53

```
gene
cistron
factor
researcher
Eeles
RA
research worker
investigator
radium
Ra
atomic number 88
```

Gambar 6.53: Synset Extraction Test Case 18

Synset ini kemudian akan diukur tingkat kemiripannya dengna *list Property RDF* dan akan diekstrak *property* yang memiliki nilai kecocokan yang tinggi. Hasil dari proses ini ditunjukkan pada Gambar 6.54

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
pubmed:researcher
```

Gambar 6.54: Hasil Property Text Similarity Question Test Case 18

Candidate Property yang didapatkan hanya pubmed:researcher saja, padahal seharusnya karena menanyakan mengenai mapped gene, maka seharusnya property gama:mapped, harus ikut terkstrak agar menjawab pertanyaan. Hal ini akan memberi pengaruh pada hasil query. SPARQL Query ditunjukkan pada *code* 6.35. Begitu juga dengan *Test Case 18*, hasil yang didapatkan sama dengan *Test Case*

17. Ketika query dijalankan , ditunjukkan pada Gambar 6.35

```
PREFIX  haz:  <http://localhost:3030/hazmap/>
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE
{
?s  rdf:type            gama: ;
    pubmed:researcher   ?researcher
    }
    LIMIT   25
```

Kode 6.35: SPARQL Query Question Test Case 18

Query yang dijalankan pada Test Case 18 sama dengan Test Case 17. Seharusnya, property gama:reported, ikut terkestrak, dikarenakan pada proses POS Tagging tidak ikut terekstrak keyword *reported* , yang hanya terekstrak hanya *keyword gene*. Hasil dari query di atas, ditunjukkan pada *code* 6.36. Karena query yang dijalankan sama dengan *Test Case 17*, maka hasil yang dikeluarkan juga sama. Sehingga dapat disimpulkan baik pertanyaan *Test Case 17 dan 18* tidak menghasilkan jawaban yang tepat.


```
{
  "head": {
    "vars": [ "s" , "researcher" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/20" } ,
        "researcher": { "type": "literal" , "value": "Berndt␣SI" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/287" } ,
        "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/237" } ,
        "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/113" } ,
        "researcher": { "type": "literal" , "value": "Cheng␣I" }
```

```
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/14" } ,
  "researcher": { "type": "literal" , "value": "Berndt␣SI" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/276" } ,
  "researcher": { "type": "literal" , "value": "Thomas␣G" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/232" } ,
  "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/286" } ,
  "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/289" } ,
  "researcher": { "type": "literal" , "value": "Yeager␣M" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/66" } ,
  "researcher": { "type": "literal" , "value": "Knipe␣DW" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/226" } ,
  "researcher": { "type": "literal" , "value": "Takata␣R" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/258" } ,
  "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/238" } ,
  "researcher": { "type": "literal" , "value": "Eeles␣RA" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/260" } ,
  "researcher": { "type": "literal" , "value": "Eeles␣RA" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/284" } ,
  "researcher": { "type": "literal" , "value": "Gudmundsson␣J" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/182" } ,
  "researcher": { "type": "literal" , "value": "Schumacher␣FR" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/68" } ,
  "researcher": { "type": "literal" , "value": "Knipe␣DW" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/227" } ,
  "researcher": { "type": "literal" , "value": "Takata␣R" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/270" } ,
  "researcher": { "type": "literal" , "value": "Eeles␣RA" }
} ,
{
  "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/193" } ,
  "researcher": { "type": "literal" , "value": "Schumacher␣FR" }
```

```
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/278" } ,
        "researcher": { "type": "literal" , "value": "Thomas␣G" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/78" } ,
        "researcher": { "type": "literal" , "value": "Lange␣EM" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/19" } ,
        "researcher": { "type": "literal" , "value": "Berndt␣SI" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/184" } ,
        "researcher": { "type": "literal" , "value": "Schumacher␣FR" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/288" } ,
        "researcher": { "type": "literal" , "value": "Yeager␣M" }
      }
    ]
  }
}
```


Kode 6.36: Hasil JSON SPARQL Query Test Case 18

Question Test Case 19:
Question : What is allele studied by researcher Schumacher?

Hasil dari POS Tagging Process pada Test Case 19 adalah sebagai berikut yang ditunjukkan Gambar 6.55:

```
What do you want to ask about?
What is allele studied by researcher Schumacher?
The following keyword have been found : allele
The following keyword have been found : researcher
The following keyword have been found : Schumacher
```

Gambar 6.55: POS Tagging Test Case 19

Dari pertanyaan yang diberikan, keyword yang didapatkan adalah *allele researcher* dan *Schumacher*. Keyword ini kemudian akan diperluas makna semantiknya menggunakan kamus Wordnet, sehing-

ga hasilnya ditunjukkan pada Gambar 6.56

```
SYNONIM WORD ARE LISTED BELOW
Synset Extraction Process is taken: 0.014219236 second
Synset Extraction Process is taken: 2.67672E-4 second
allele
allelomorph
researcher
Schumacher
research worker
investigator
```

Gambar 6.56: Synset Extraction Test Case 19

Dari Synset atau *Synonym Set* yang didapatkan adalah *allele, allelemorph, researcher, Schumacher, researcher worker, investigator*. Dengan padanan kata yang didapatkan, maka akan dicocokan kemiripan string dengan list property. Proses ini adalah melakukan pengecekan secara sintatik. Hasil property yang didapatkan ditunjukkan pada Gambar 6.57

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
snp:allele
pubmed:researcher
```

Gambar 6.57: Hasil Property Text Similarity Question Test Case 19

Dari pengujian dengan pertanyaan *test case* 19 didapatkan jika *candidate property* yang berhubungan dengan pertanyaan tersebut adalah *snp:allele* dan *pubmed:researcher*. Sehingga dengan candidate property tersebut, nilainya akan menjadi parameter di slot yang sudah ada di SPARQL Template. Code untuk SPARQL ditunjukkan pada *code* 6.37

```
PREFIX haz: <http://localhost:3030/hazmap/>
```

```
PREFIX  che:  <http://localhost:3030/che/>
PREFIX  owl:  <http://www.w3.org/2002/07/owl#>
PREFIX  xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw:  <http://localhost:3030/biw/>
PREFIX  ctd:  <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp:  <http://localhost:3030/snpedia/>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE   {
?s      rdf:type            snp: ;
        owl:sameAs          ?same ;
        snp:allele          ?p .
        ?same   pubmed:researcher   ?o
        FILTER regex(?o, "Schumacher")
        } LIMIT   25
```

Kode 6.37: SPARQL Query Question Test Case 19

Hasil *code* SPARQL untuk *test case* 19 ditunjukkan pada *JSON code* 6.38

```
{
  "head": {
    "vars": [ "s" , "same" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/4" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/184" } ,
        "p": { "type": "literal" , "value": "(G;G)" } ,
        "o": { "type": "literal" , "value": "Schumacher_FR" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/12" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/184" } ,
        "p": { "type": "literal" , "value": "(G;T)" } ,
        "o": { "type": "literal" , "value": "Schumacher_FR" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/13" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/182" } ,
        "p": { "type": "literal" , "value": "(G;G)" } ,
        "o": { "type": "literal" , "value": "Schumacher_FR" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/snpedia/33" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/193" } ,
        "p": { "type": "literal" , "value": "(T;T)" } ,
        "o": { "type": "literal" , "value": "Schumacher_FR" }
      }
    ]
  }
}
```

Kode 6.38: Hasil JSON SPARQL Query Test Case 19

Question Test Case 20:
Question : What is the causes of Prostate Cancer researched in journal Nat?

Hasil dari POS Tagging Process pada Test Case 20 adalah sebagai berikut yang ditunjukkan Gambar 6.58:

```
What do you want to ask about?
What is the causes of Prostate Cancer researched in journal Nat?
The following keyword have been found : causes
The following keyword have been found : Prostate
The following keyword have been found : Cancer
The following keyword have been found : journal
The following keyword have been found : Nat
```

Gambar 6.58: POS Tagging Test Case 20

Hasil POSTAG kemudian akan mengekstrak keyword, *causes, Prostate, Cancer, journal, Nat*. Keyword ini kemudian akan diperluas makna semantiknya dengan menggunakan kamus *Wordnet*. Hasilnya ditunjukkan pada Gambar 6.59

Synset yang dihasilkan pada Gambar 6.59 merupakan hasil perluasan makna semantik dengan menggunakan *Wordnet*. Kumpulan kata pada Gambar 6.59 kemudian akan dicocokan secara sintatik dengan menggunakan Levenshtein pada kumpulan property RDF. Hasil nya akan berupa *candidate property* yang ditunjukkan pada Gambar 6.60

Karena *candidate property* yang didapatkan sudah mencukupi yaitu 2 property yang dibutuhkan untuk menjawab pertanyaan, maka property ini kemudian akan diteruskan ke SPARQL Template. Hasil *code* SPARQL ditunjukkan pada *code* 6.39

```
PREFIX haz: <http://localhost:3030/hazmap/>
PREFIX che: <http://localhost:3030/che/>
```

```
SYNONIM WORD ARE LISTED BELOW
Synset Extraction Process is taken: 0.001194729 second
Synset Extraction Process is taken: 6.9483E-5 second
Synset Extraction Process is taken: 9.653E-5 second
Synset Extraction Process is taken: 1.1938E-4 second
cause
campaign
crusade
drive
movement
effort
causes
Prostate
Cancer
journal
Nat
prostate gland
prostate
cancer
malignant neoplastic disease
Cancer the Crab
Crab
diary
```

Gambar 6.59: Synset Extraction Test Case 20

```
PREFIX  owl: <http://www.w3.org/2002/07/owl#>
PREFIX  xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX  rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  biw: <http://localhost:3030/biw/>
PREFIX  ctd: <http://localhost:3030/ctd/>
PREFIX  pubmed: <http://bio2rdf.org/bio2rdf_vocabulary:>
PREFIX  snp: <http://localhost:3030/snpedia/>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  gama: <http://localhost:3030/gama/>
SELECT  * WHERE  {
    ?s      rdf:type          che: ;
            owl:sameAs        ?same ;
            che:causes        ?p .
            ?same   pubmed:journal   ?o
            FILTER regex(?o, "Nat")
            }
            LIMIT   25
```

Kode 6.39: SPARQL Query Question Test Case 20

Dari hasil *code* SPARQL pada *code* 6.39, dihasilkan JSON pada *code* 6.40

```
{
```

```
####################################
THE PROPERTY IS RETRIEVED BELOW :
####################################
che:causes
pubmed:journal
```

Gambar 6.60: Hasil Property Text Similarity Question Test Case 20


```
"head": {
  "vars": [ "s" , "same" , "p" , "o" ]
} ,
"results": {
  "bindings": [
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/20" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Commun" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/287" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/237" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/14" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Commun" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/276" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/232" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/286" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
```

```
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/289" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/226" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/258" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/238" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/260" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/284" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/227" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/270" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/278" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Genet" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/19" } ,
      "p": { "type": "literal" , "value": "Agent␣Orange" } ,
      "o": { "type": "literal" , "value": "Nat␣Commun" }
    } ,
    {
      "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
      "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/288" } ,
```

```
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/272" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/261" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/2" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Commun" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/274" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/234" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/285" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://127.0.0.1:3333/che/1" } ,
        "same": { "type": "uri" , "value": "http://127.0.0.1:3333/gama/262" } ,
        "p": { "type": "literal" , "value": "Agent␣Orange" } ,
        "o": { "type": "literal" , "value": "Nat␣Genet" }
      }
    ]
  }
}
```


Kode 6.40: Hasil JSON SPARQL Query Test Case 20

Rangkuman dari keseluruhan pengujian ditunjukkan pada tabel 6.1, dari tabel tersebut, diketahui bahwa dari 18 soal pengujian, yang sukses dieksekusi dan tepat benar adalah sebanyak 16 pertanyaan, 2 pertanyaan sukses dieksekusi namun salah.

Tabel 6.1: Tabel Keseluruhan Pengujian

No	Test Case	Result
1	What is allele of variance Rs1800896?	Success and Right
2	How much is the magnitude of variance Rs4242382	Success and Right
3	What is the repute of variance Rs4242382	Success and Right
4	What is the summary of variance Rs4792311?	Success and Right
5	What are the causes of disease Prostate Cancer?	Success and Right
6	What is the evidence of the causes of methyl bromide?	Success and Right
7	What is the symbol of chemical pyrimidine?	Success and Right
8	What are the interactions of chemical tetrachlorobiphenyl?	Success and Right
9	What is the synonyms of chemical chlorophenoxybutyric?	Success and Right
10	What is the formula of chemical chlorophenoxybutyric?	Success and Right
11	What is the category of chemical Metaldehyde?	Success and Right
12	What is the source of chemical Metaldehyde?	Success and Right
13	What is the cas number of chemical propylphthalate?	Success and Right
14	Who is the researcher that has published journal Nat Genet?	Success and Right
15	When is publication of journal Nat Genet?	Success and Right

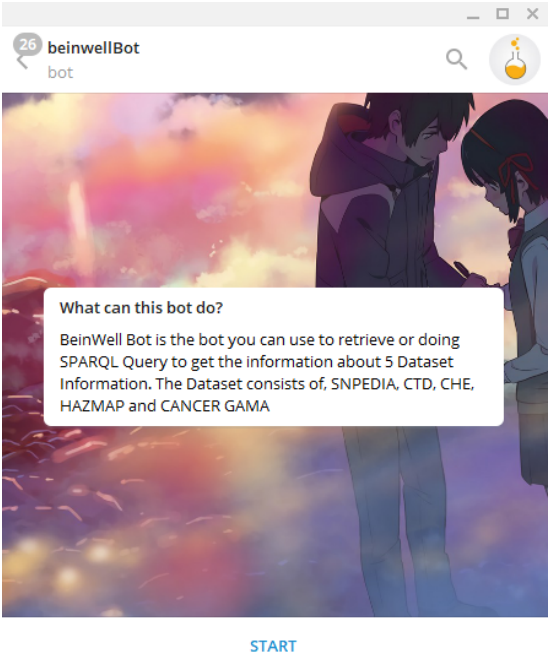
16	What is the region of researcher Eeles RA found ?	Success and Right
17	What is the mapped gene of researcher Eeles RA found ?	Success and False
18	What is the reported gene of researcher Eeles RA found ?	Success and False
19	What is allele studied by researcher Schumacher?	Success and Right
20	What is the causes of Prostate Cancer researched in journal Nat?	Success and Right

Kemudian setelah pengujian functional pada *core system* pada arsitektur java dengan menggunakan *Question Test Case*, maka selanjutnya adalah melakukan pengujian functional kepada Chat bot Telegram, yaitu dengan mencoba semua functional serta menguji dengan menginputkan pertanyaan pada Chat bot telegram

Berikut adalah tampilan awal ketika Chat bot pertama kali di-*add*, seperti pada gambar 6.61

Kemudian ketika memasukkan perintah *snpedia sample*, chatbot dapat merespond dan memberikan balasan, seperti gambar 6.62

Ketika memasukkan perintah *ctd sample*, bot juga berhasil memberikan respond terhadap request, ditunjukkan pada gambar 6.63 begitu juga dengan perintah *che sample* dan *haz sample* yang ditunjukkan pada gambar 6.64 dan 6.65

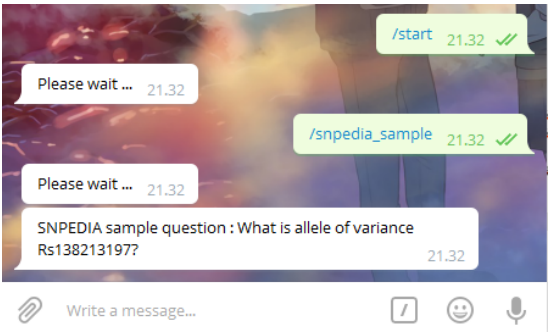


Gambar 6.61: Tampilan awal Chat bot Telegram beinwellBot

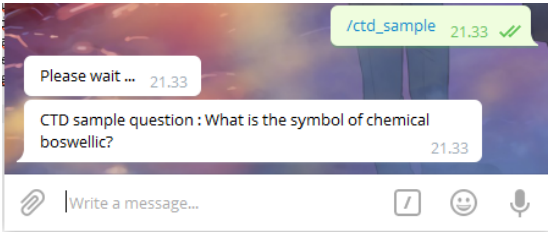
6.1.2 Pengujian Performa

Pada pengujian *perf* dilakukan dengan cara menghitung waktu respond system pada setiap proses terhadap setiap pertanyaan yang telah diberikan. Proses-proses yang akan dinilai kecepatan prosesnya adalah sebagai berikut:

- Proses POS Tagging
- Synset Extraction
- Levenshtein Distance
- SPARQL Query Process



Gambar 6.62: Hasil eksekusi perintah snpedia sample



Gambar 6.63: Hasil eksekusi perintah ctd sample

Berikut adalah hasil pengukuran kecepatan waktu proses, di tiap tahapannya, ditunjukkan pada tabel 6.2

Tabel 6.2: Tabel Kecepatan Proses Uji 1

Test Case	POS Tagging (detik)	Wordnet Extraction (detik)	Levenshtein Distance (detik)	SPARQL Execution (detik)
1	0,0718140	0,0872966	0,0325940	4
2	0,0778770	0,0210780	0,0389449	7
3	0,0685500	0,0715340	0,0336940	3

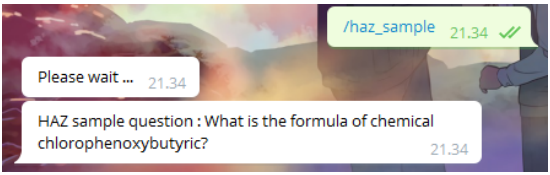
4	0,0918670	0,0990014	0,0256280	3
5	0,0918670	0,0778770	0,0442069	8
6	0,0152023	0,0951308	0,0456334	4
7	0,0147359	0,0433680	0,0200348	27
8	0,0741460	0,0923795	0,0263722	9
9	0,0120779	0,0675240	0,0288223	3
10	0,0937320	0,0802090	0,0528774	11
11	0,0774100	0,0286325	0,0248361	3
12	0,0699490	0,0361404	0,0264767	3
13	0,0830060	0,0999340	0,0468967	12
14	0,0853380	0,0732130	0,0336003	4
15	0,0360937	0,0568920	0,0350832	3
16	0,0834730	0,0634200	0,0322867	15
17	0,0955970	0,0439281	0,0817831	9
18	0,0974630	0,0690170	0,0368571	15
19	0,0186065	0,0893482	0,0200511	2
20	0,010166	0,0851046	0,0435092	5

Berdasarkan tabel 6.2, menunjukkan mengenai pemrosesan pada setiap tahapan di sistem konversi. Deskriptif dari tabel 6.2 ditunjukkan pada tabel 6.3

Tabel 6.3: Tabel Deskriptif Pengujian 1

	POS Tagging (detik)	Wordnet Extraction (detik)	Levenshtein Distance (detik)	SPARQL Execution (detik)
MAX	0,0974630	0,0999340	0,0817831	27
MIN	0,0120779	0,0210780	0,0200348	3
AVERAGE	0,0688999	0,0670320	0,0370349	7,9444444

Gambar 6.64: Hasil eksekusi perintah che sample



Gambar 6.65: Hasil eksekusi perintah haz sample

Dari tabel 6.3 diketahui bahwa proses yang paling lama memakan waktu adalah *SPARQL Execution* sedangkan proses yang paling cepat adalah proses *POS Tagging*. Rata-rata kecepatan dalam pengujian 1 ini adalah mencapai 8 detik, untuk menjawab sebuah pertanyaan dengan batasan 25 *instances*

Tabel 6.4: Tabel Kecepatan Proses Uji 2

Test Case	POS Tagging (detik)	Wordnet Extraction (detik)	Levenshtein Distance (detik)	SPARQL Execution (detik)
1	0,078343	0,0343217	0,030438138	9
2	0,023363	0,0323165	0,039067065	3
3	0,067618	0,0837058	0,030574772	2
4	0,092333	0,0909805	0,029388901	2
5	0,0122178	0,0892551	0,049787471	3
6	0,0164613	0,064819	0,049211557	3

7	0,0141763	0,0274201	0,019470112	3
8	0,0138965	0,0831462	0,022247093	3
9	0,0791358	0,0791358	0,030529538	2
10	0,073213	0,05596	0,037768342	2
11	0,074146	0,0974158	0,027060993	2
12	0,071814	0,0110986	0,033661394	8
13	0,083473	0,0691097	0,071649844	3
14	0,0104924	0,094198	0,036569414	3
15	0,080674	0,060622	0,042155087	3
16	0,087203	0,058758	0,039271783	3
17	0,0103525	0,0793689	0,033870776	2
18	0,0144562	0,061555	0,038425398	3
19	0,0158085	0,0297983	0,0225730	2
20	0,092799	0,097928	0,0417730	2

Berdasarkan tabel 6.4, menunjukkan waktu proses sistem konversi per tahapnya, pada percobaan ke-2. Deskriptif dari Tabel tersebut ditunjukkan pada tabel 6.5

Tabel 6.5: Tabel Deskriptif Pengujian 2

	POS Tagging (detik)	Wordnet Extraction (detik)	Levenshtein Distance (detik)	SPARQL Execution (detik)
MAX	0,092333	0,0974158	0,071649844	9
MIN	0,0103525	0,0110986	0,019470112	2
AVERAGE	0,050187156	0,065177039	0,036730427	3,277777778

Berdasarkan hasil percobaan ke-2, didapatkan jika waktu proses terlama adalah SPARQL dengan waktu proses 9 detik, di mana pro-

ses Levenshtein Distance merupakan proses tercepat. Jika dilihat secara keseluruhan, total proses yang dibutuhkan oleh sistem untuk merespon dan mengolah pertanyaan adalah 3,4 detik.

Tabel 6.6: Tabel Kecepatan Proses Uji 3

Test Case	POS Tagging (detik)	Wordnet Extraction (detik)	Levenshtein Distance (detik)	SPARQL Execution (detik)
1	0,085805	0,0806281	0,028196035	3
2	0,099794	0,094664	0,024866455	3
3	0,071815	0,0144095	0,028158262	2
4	0,071815	0,0335756	0,026948142	3
5	0,086737	0,073679	0,043207122	3
6	0,0214044	0,053161	0,043463602	3
7	0,0152955	0,0306844	0,020352404	3
8	0,0149225	0,070089	0,021100394	3
9	0,07368	0,0741461	0,026598396	3
10	0,080208	0,054094	0,036438376	2
11	0,074612	0,0368865	0,02848236	2
12	0,087203	0,0900020	0,040609677	3
13	0,085804	0,0772238	0,034292336	3
14	0,084405	0,0775503	0,042004463	2
15	0,088602	0,062021	0,039723188	3
16	0,085338	0,05969	0,030488035	3
17	0,0134303	0,0859442	0,041470518	3
18	0,0163681	0,080208	0,037652694	3
19	0,0172541	0,0309641	0,0424763	2
20	0,088136	0,071815	0,0411155	2

Pada tabel 6.6 menunjukkan hasil pengujian *test case* ke-3. Dari ta-

bel tersebut kemudian, dicari deskriptifnya, ditunjukkan pada tabel 6.7

Tabel 6.7: Tabel Deskriptif Pengujian 3

	POS Tagging (detik)	Wordnet Extraction (detik)	Levenshtein Distance (detik)	SPARQL Execution (detik)
MAX	0,099794	0,094664	0,043463602	3
MIN	0,0134303	0,0144095	0,020352404	2
AVERAGE	0,064291044	0,06381425	0,033002914	2,777777778

Dari tabel 6.7 menunjukkan jika secara keseluruhan, pada pengujian ke-3, proses paling lama tetap pada SPARQL Execution, namun pada pengujian ke-3 ini, lebih stabil pada angka 3 detik, di mana rata-ratanya mencapai 2,7 detik. Secara keseluruhan, pada pengujian ke-3 ini, keseluruhan waktu yang diperlukan untuk merespon adalah 2,9 detik.

Dengan mempertimbangkan pengujian 1 hingga pengujian 3, maka dicari rata-rata keseluruhan, seperti ditunjukkan pada tabel 6.8

Tabel 6.8: Hasil Rata-rata Pengujian 1 - 3

Test Case	POS	Synset	Levens	SPARQL	Total
1	0,0787	0,0674	0,0304	5,3333	5,5098
2	0,0670	0,0494	0,0343	4,3333	4,4840
3	0,0693	0,0565	0,0308	2,3333	2,4900
4	0,0853	0,0745	0,0273	2,6667	2,8538
5	0,0636	0,0803	0,0457	4,6667	4,8563
6	0,0177	0,0710	0,0461	3,3333	3,4682

7	0,0147	0,0338	0,0200	11,0000	11,0685
8	0,0343	0,0819	0,0232	5,0000	5,1394
9	0,0550	0,0736	0,0287	2,6667	2,8239
10	0,0824	0,0634	0,0424	5,0000	5,1882
11	0,0754	0,0543	0,0268	2,3333	2,4898
12	0,0763	0,0457	0,0336	4,6667	4,8223
13	0,0841	0,0821	0,0509	6,0000	6,2171
14	0,0601	0,0817	0,0374	3,0000	3,1791
15	0,0685	0,0598	0,0390	3,0000	3,1673
16	0,0853	0,0606	0,0340	7,0000	7,1800
17	0,0398	0,0697	0,0524	4,6667	4,8286
18	0,0428	0,0703	0,0376	7,0000	7,1507
19	0,0172	0,0500	0,0284	2,0000	2,0956
20	0,0637	0,0849	0,0421	3,0000	3,1908

Dari tabel 6.8 didapatkan nilai rata-rata per proses dari pengujian 1 - 3, serta nilai total secara keseluruhan untuk merespon pertanyaan. Didapatkan jika, rata-rata sistem untuk merespon pertanyaan adalah sekitar 4,82 detik

6.2 Pembahasan

Pada subbab ini akan dibahas dan disimpulkan hasil dari pengujian fungsional dan non-fungsional dari perangkat lunak.

- Pengujian Integrasi
  Pada pengujian integrasi perangkat lunak telah dilakukan beberapa skenario yang ditujukan pada sistem utama yang dibangun di arsitektur java, baik untuk menguji keseluruhan kinerja tahapan dan juga ketepatan hasil yang dihasilkan. Ske-

nario yang dibuat adalah *Question Test Case*, di mana sistem akan diberikan pertanyaan yang tersebar di 5 dataset yang digunakan. Dari 20 skenario yang diberikan sistem dapat menjawab 18 pertanyaan dengan tepat pertanyaan yang diberikan, namun terdapat 2 pertanyaan yang sukses mengeluarkan jawaban, namun keduanya salah. Sehingga *precission* dari sistem sendiri mencapai 90 persen. Di samping pengujian dengan terhadap sistem utama, pengujian fungsional berikutnya adalah dengan menguji Chat bot Telegram beinwellBot. Chat bot yang sudah dikembangkan kemudian akan diuji coba terhadap semua command yang ada pada Chat bot. Command yang diujikan dapat dijalankan tanpa masalah. Chat bot juga dapat terintegrasi dengan baik dengan *core system*. Hal ini ditunjukkan dengan Chat bot yang mampu menjawab pertanyaan yang sama yang diajukan pada *core system*nya

- Pengujian Performa
  Pada pengujian performa dilakukan dengan mengukur kecepatan proses yang dihasilkan pada setiap tahap yang ada pada sistem untuk setiap *Question Test Case yang diberikan*. Pengujian dilakukan sebanyak 3 kali. Secara keseluruhan proses yang paling lama memakan waktu adalah proses SPARQL Execution, di mana dari pengujian 1 - 3, rata-rata membutuhkan waktu selama 4,6 detik. Proses yang paling cepat pengolahannya adalah proses *Levenshtein Distance* dengan rata-rata total keseluruhan adalah selama 0,03 detik. Waktu total keseluruhan yang dibutuhkan untuk merespon sebuah pertanyaan sendiri adalah sekitar 4,8 detik

BAB 7

KESIMPULAN DAN SARAN

Pada bab ini akan dijelaskan kesimpulan dan saran dalam pengerjaan tugas akhir.

7.1 Kesimpulan

Berdasarkan dengan pengerjaan tugas akhir dengan judul ”Question Answering System dengan Pendekatan Template Based Berbasis Linked Data pada Data Kanker Prostat” yang telah dilakukan dapat disimpulkan beberapa hal sebagai berikut:

1. Question Answering System yang dibangun dengan menggunakan pendekatan Template Based serta dilakukan integrasi dengan menggunakan Chat Bot Telegram. Question Answering System yang dibangun dengan menggunakan Template Based memanfaatkan teknologi NLP seperti POS Tagger dan Wordnet untuk melakukan proses secara semantik dan Levenshtein untuk memproses secara sintatik
2. Question Answering System yang dibangun pada platform Telegram dapat menjawab Test Case Question dengan struktur bahasa sederhana yang dibuat. Dari 20 Question Test Case, 18 di antaranya dapat dieksekusi dengan benar dan menghasilkan jawaban yang tepat
3. Pertanyaan yang dijawab masih berupa pertanyaan yang dibangun secara sintatik
4. Proses SPARQL Query memakan waktu yang paling lama dari semua proses pada sistem konversi, di mana paling lama mencapai 11 detik dan paling cepat 2 detik.

7.2 Saran

Saran penulis untuk penelitian selanjutnya sebagai berikut:

1. Pada penelitian ini perangkat lunak yang dikembangkan hanya berbasis *Chat bot Telegram*. Kedepannya peneliti berharap dapat dikembangkan ke dalam *platform* lain seperti android. Dengan dikembangkannya perangkat lunak berbasis android, pengguna dapat dengan mudah melakukan tanya jawab seputar kanker prostat.
2. Pada penelitian ini, proses NLP yang digunakan adalah POS Tagging, Wordnet dan Levenshtein. Namun tidak menutup kemungkinan jika proses NLP lain seperti Named Entity Recognition serta Word2Vec dapat digunakan untuk melakukan ekstraksi kata keyword yang luas dan lebih memahami konteks pertanyaan, sehingga struktur pertanyaan dapat berkembang cukup kompleks
3. Pada penelitian ini belum dilakukan *user acceptance test*. Pada penelitian selanjutnya diharapkan melakukan *user acceptance test* untuk menguji kemudahan pengguna dalam menggunakan aplikasi.
4. Untuk mengimplementasikan Linked Data dengan NLP untuk mengembangkan sebuah Question Answer System, harus dipastikan bahwa property yang digunakan pada RDF memiliki konsistensi dan bermakna, tidak menggunakan singkatan sehingga mudah dipahami oleh orang awam
5. Pengembangan lebih lanjut Question Answering System agar dapat mendukung pertanyaan dengan struktur kalimat yang lebih kompleks dan tidak terlalu *syntatix*

DAFTAR PUSTAKA

[1] About WordNet - WordNet - About WordNet.

[2] Apache Jena - Fuseki: serving RDF data over HTTP. Accessed February 8th, 2017, https://jena.apache.org/documentation/serving_data/.

[3] Apache jena - semantic web standards. Accessed February 8th, 2017, https://www.w3.org/2001/sw/wiki/Apache_Jena.

[4] Keyword-based versus natural-language search – Inbenta. Accessed February 23th, 2017, https://www.inbenta.com/en/blog/keyword-based-versus-natural-language-search/.

[5] Levenshtein Distance.

[6] Penn Treebank P.O.S. Tags. Accessed February 8th, 2017, https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html.

[7] The Stanford Natural Language Processing Group. Accessed February 8th, 2017, http://nlp.stanford.edu/software/tagger.shtml.

[8] What are the differences between semantic based search and keyword based search? - Quora. Accessed January 16th, 2017, https://www.quora.com/What-are-the-differences-between-semantic-based-search-and-keyword-b

[9] Ahmed Al-Nazer, Saeed Albukhitan, and Tarek Helmy. Cross-domain semantic web model for understanding multilingual natural language queries: English/arabic health/food domain use case. *Procedia Computer Science*, 83:607–614, 2016.

[10] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[11] Amit Mishra and Sanjay Kumar Jain. A survey on question answering systems with classification. *Journal of King Saud University - Computer and Information Sciences*, 28(3):345 – 361, 2016.

[12] Ajitkumar M Pundge, SA Khillare, and C Namrata Mahender. Question answering system, approaches and techniques: A review. 2016.

[13] A. Shah, J. Pareek, H. Patel, and N. Panchal. Nlkbidb - natural language and keyword based interface to database. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1569–1576, Aug 2013.

[14] Eriks Sneiders. *Automated Question Answering Using Question Templates That Cover the Conceptual Model of the Database*, pages 235–239. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[15] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web*, pages 639–648. ACM, 2012.

LAMPIRAN A

DAFTAR PRODUK

Halaman ini sengaja dikosongkan

UCAPAN TERIMA KASIH

Dalam pembuatann tugas akhir ini tidak terlepas dari dukungan serta doa dari berbagai pihak. Oleh karena itu dalam kesempatan ini penulis menyampaikan terimakasih kepada:

1. Ibu Nur Aini Rakhmawati, S.Kom., M.Sc., Eng. selaku dosen pembimbing penulis yang telah memberikan ide, bimbingan, saran, kritik, ilmu, dan pengalamannya yang sangat bermanfaat sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Seluruh dosen Jurusan Sistem Informasi ITS yang telah memberikan ilmu pengetahuan dan pengalaman yang sangat berharga dan bermanfaat bagi penulis.
3. Seluruh keluarga besar saya khususnya Bapak, Ibu serta Kakak dan Adik saya yang telah memberikan semangat serta kepercayaan kepada saya dalam mengerjakan tugas akhir
4. Teman-teman dan Sahabat terdekat saya angkatan 2013 (13ELTRANIS) yang senantiasa menemani daam prosses pengerjaan serta diskusi yang diberikan.
5. Sahabat terdekat saya, SAHABAT SAMBAT, yang menjadi tempat curahan hati dan cerita selama pengerjaan Tugas Akhir
6. Rekan-rekan anggota Laboratorium Akuisisi Data dan Diseminasi Informasi atas segala pencerahan, inspirasi dan bantuan yang diberikan
7. Rekan-rekan organisasi Himpunan Mahasiswa Sistem Informasi HMSI ITS yang telah memberikan pengalaman, pelajaran berharga dan bermanfaat selama disana.
8. Serta seluruh pihak-pihak lain yang tidak dapat disebutkan satu per satu yang telah banyak membantu penulis selama perkuliahan hingga dapat menyelesaikan tugas akhir ini.

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis lahir di Kediri pada tanggal 7 Desember 1994. Merupakan anak kedua dari 3 bersaudara dan telah menempuh pendidikan formal yaitu: SD Katolik St. Yustinus De Yacobis Krian, SMP Negeri 2 Krian, dan SMA Negeri 1 Sidoarjo.

Pada tahun 2013 melanjutkan pendidikan ke tingkat perguruan tinggi di Departemen Sistem Informasi FTIF - Institut Teknologi Sepuluh Nopember dan terdaftar sebagai mahasiswa dengan NRP 5213100078. Selama menjadi mahasiswa, penulis banyak terlibat aktif dalam kegiatan baik kegiatan internal dan eksternal. Penulis mengikuti organisasi kemahasiswaan, Himpunan Mahasiswa Sistem Informasi 2014/2015 sebagai salah satu staf di Departemen Riset dan Teknologi, pada tahun kepengurusan 2015/2016, penulis memegang peranan untuk menjadi Kepala Divisi Application Development di Himpunan Sistem Informasi, Departemen Aplikasi dan Teknologi. Di samping aktif dalam organisasi kemahasiswaan, penulis juga aktif dalam kegiatan eksternal, dengan menjadi Student Ambassador Tokopedia Chapter Surabaya Batch 2 selama 1 tahun kepengurusan. Penulis juga pernah mendapat beberapa penghargaan di tingkat Nasional, yaitu menjadi Juara 1 dalam ajang AMIKOM ICT Award 2016 dan menjadi Finalis di ajang Gemastik 9 di Universitas Indonesia. Penulis juga aktif menjadi asisten dosen pada mata kuliah Interaksi Manusia dan Komputer dan Perencanaan Sumber Daya Perusahaan.

Pada tahun keempat, penulis mulai menekuni pada bidang Data Science, Natural Language Processing, maka penulis mengambil bidang minat Laboratorium Akuisisi Data dan Diseminasi Informasi (ADDI). Penulis dapat dihubungi melalui email: herjunoniko@gmail.com.