PENGEMBANGAN APLIKASI KONVERSI OTOMATIS DATA *JSON* KE *OWL ONTOLOGY*

Disusun Oleh:

N a m a : Yaya Rizqi Isnanda

NIM : 14523131

PROGRAM STUDI TEKNIK INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2018

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGEMBANGAN APLIKASI KONVERSI OTOMATIS DATA *JSON* KE *OWL ONTOLOGY*

TUGAS AKHIR

Disusun Oleh:

N a m a : Yaya Rizqi Isnanda
NIM : 14523131

Yogyakarta, 26 Oktober 2018

Pembimbing,

(Dhomas Hatta Fudholi S.T., M.Eng., Ph.D.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGEMBANGAN APLIKASI KONVERSI OTOMATIS DATA *JSON* KE *OWL ONTOLOGY*

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Teknik Informatika di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 19 Oktober 2018

Tim Penguji

Dhomas Hatta Fudholi S.T., M.Eng., Ph.D. _____

Anggota 1

Chanifah Indah Ratnasari S.Kom., M.Kom. _____

Anggota 2

Hanson Prihantoro Putro S.T., M.T. _____

Mengetahui,

Ketua Program Studi Teknik Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Yaya Rizqi Isnanda

NIM : 14523131

Tugas akhir dengan judul:

PENGEMBANGAN APLIKASI KONVERSI OTOMATIS DATA *JSON* KE *OWL ONTOLOGY*

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 3 Oktober 2018

(Yaya Rizqi Isnanda)

HALAMAN PERSEMBAHAN

Dengan mengucap syukur *Alhamdulillah,* saya persembahkan karya ini untuk orang – orang yang saya sayangi:

Kedua orang tua tercinta,
Ayahanda Suharno dan **Ibunda Ismaryati**

Motivator terbesar dalam hidupku yang selalu memberkan doa, semangat, nasehat dan motivasi demi kelancaran semuanya. Semoga dengan karya ini dapat membuat bangga kedua orang tua tercinta, *Aamiin.*

Kakak dan adik perempuan tercinta,
Lala isna hasni dan **Rara isna Al-jannah**

Saudara sekaligus teman yang selalu memberikan semangat dan bersedia menjadi tempat untuk bercerita. Semoga tujuan kita membahagiakan kedua orang tua kita dan orang yang kita cintai tercapai, *aamiin.*

HALAMAN MOTO

"*Allah tidak akan membebani seorang manusia melainkan sesuai dengan kemampuan manusia itu sendiri"*

(Q.S. Al Baqarah: 286)

"Karena sesungguhnya sesudah kesulitan itu ada kemudahan, sesudah kesulitan itu ada kemudahan"

(QS. Al Insyirah: 5-6)

"Sedikit pengetahuan yang diterapkan jauh lebih berharga ketimbang banyak pengetahuan yang tak dimanfaatkan"

(Khahlil Gibran).

"Ada dua kenikmatan yang banyak manusia tertipu,yaitu nikmat sehat dan waktu senggang"

(HR. Bukhari no. 6412, dariIbn'Abbas)

"Usaha keras itu tak akan mengkhianati, kalau mengkhianati berarti usahanya belum keras"

(Melody Nurramdhani Laksani)

KATA PENGANTAR

Assalamu'alaikum warahmatullahi wabarakatuh.

Alhamdulillahirabbil'alamiin, puji syukur atas kehadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya kepada kita semua. Sholawat serta salam semoga selalu tercurah kepada junjungan Nabi Muhammad SAW beserta keluarga dan sahabatnya sehingga penulis dapat menyelesaikan laporan Tugas Akhir yang berjudul "Sistem Informasi Eksekutif Monitoring Akademik Mahasiswa" dengan baik dan lancar.

Laporan Tugas Akhir ini disusun sebagai salah satu syarat yang harus dipenuhi untuk menyelesaikan pendidikan pada jenjang Strata 1 (S1) Jurusan Teknik Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia. Penulis menyadari bahwa dalam penulisan dan penyusunan laporan ini tidak terlepas dari bantuan, bimbingan, serta dukungan dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih sebesar-besarnya kepada:

1. Bapak Fathul Wahid, S.T., M.Sc., Ph.D., selaku Rektor Universitas Islam Indonesia.
2. Bapak Prof. Dr. Ir. Hari Purnomo, M.T., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
3. Bapak Hendrik , S.T., M.Eng., selaku Ketua Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Bapak Dr. Raden Teduh Dirgahayu, S.T., M.Sc., selaku Ketua Program studi S1 Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
5. Bapak Dhomas Hatta Fudholi S.T., M.Eng., Ph.D selaku dosen pembimbing tugas akhir yang telah sabar dalam memberikan ilmu dan saran yang sangat bermanfaat serta meluangkan waktunya untuk membimbing penulis dalam menyelesaikan tugas akhir ini.
6. Bapak dan Ibu Dosen Jurusan Teknik Informatika yang telah memberikan ilmunya kepada penulis.
7. Orang tua tercinta Bapak Suharno dan Ibu Ismaryati, serta kedua saudara tercinta Lala Isna Hasni dan Rara Isna Al-Jannah, atas segala dukungan, arahan, pengorbanan, kasih sayang, nasihat, serta doa yang tidak pernah putus kepada penulis.

8. Semua pihak yang dicintai dan disayangi penulis, yang tidak dapat disebutkan satu persatu, terima kasih atas bantuan dan doanya.

Penulis menyadari bahwa Tugas Akhir ini tidaklah sempurna dan tidak dapat terlepas dari banyaknya kekurangan karena keterbatasan kemampuan dan pengalaman penulis. Oleh karena itu, penulis mengharapkan kritik dan saran yang bersifat membangun sebagai bahan evaluasi agar dapat lebih baik lagi ke depannya. Semoga Tugas Akhir ini dapat diterima dan memberikan manfaat bagi orang-orang yang menggunakannya. Aamiin.

Wassalamu'alaikum warahmatullahi wabarakatuh

Yogyakarta, 20 September 2018

(Yaya Rizqi Isnanda)

SARI

JSON merupakan standar format dalam pertukaran data. API (*Application programming interface*) merupakan salah satu cara pertukaran data yang menggunakan format penulisan *JSON*. Salah satu keunggulan data *JSON* adalah ringan digunakan, dikarenakan penulisan format *JSON* yang sederhana. Sehingga ukuranya lebih kecil dibanding *XML*. Selain itu, keunggulan dari *JSON* yang lain adalah dapat digunakan dalam berbagai *platform*, sehingga dalam penggunaanya *JSON* merupakan format pertukaran data yang efisien. Format pertukaran data *JSON* sangat banyak digunakan untuk saat ini. *OWL ontology* dapat digunakan untuk membangun *semantic web* atau atau web semantik. Dalam membuat web semantik salah satu komponen terpenting adalah ontology/basis pengetahuan. Karena yang membedakan web semantik dan web biasa adalah adanya basis pengetahuan pada web semantik yang dapat membantu web untuk lebih memahami kebutuhan *user*. Namun untuk saat ini ketersediaan *ontology* masih sangat sedikit. Sehingga *developer* web samantik akan lebih mengeluarkan *effort* dalam membuat *ontology* untuk sistemnya. Padahal untuk dapat membuat *ontology* sendiri tidaklah mudah. Perlu serangkaian proses yang tidak setiap *developer* punya waktu atau bahkan mengerti bagaimana cara membuat ontology. Untuk itulah alasan mendasar dari penelitian terhadap pengembangan aplikasi konversi otomatis data *JSON* ke *OWL Ontology*. Dalam penelitian ini penulis mencoba membuat aplikasi yang dapat memanfaatkan data *JSON* untuk dapat dikonverikan menjadi *OWL Ontology*. Untuk membuat aplikasi ini terlebih dahulu penulis harus bisa mendapatkan *JSON* Schema dari data *JSON* yang akan dikonversikan. Lalu penulis harus melakukan analisa pada *JSON Schema* tersebut. Sehingga setelah penulis dapat menganalisa struktur *JSON Schema* dan melakukan generate ke dalam *list* yang dibutuhkan untuk membuat *OWL Onology*, penulis dapat membuat aplikasi yang secara otomatis dapat melakukan konversi data *JSON* ke *OWL Ontology*

Kata kunci: *JSON*, *list*, Ontology, konversi, aplikasi, otomatis dan bahasa indonesia.

GLOSARIUM

Dashboard	Sebuah tampilan visual dari informasi penting yang ditampilkan dalam satu layar agar dapat dilihat secara sekilas untuk mencapai satu tujuan atau lebih.
Debug	langkah untuk menelusuri kesalahan kode program
Waterfall	metode pengembangan perangkat lunak.
JSON	Salah satu format data ayng digunakan untuk standar pertukaran data
OWL	Salah satu bentuk dari ontology
Ontology	Basis pengetahuan yang digunakan dlam pengembangan web semantic
Homepage	Halaman awal sebuah web
Bookmark	Sebuah penanda dalam web
Mapping	Proses pemetakan data dengan menggunakan aturan
Parsing	Proses mendapatkan data dari sebuah file yang di terima

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

JSON merupakan standar format dalam pertukaran data. API (*Application programming interface*) merupakan salah satu cara pertukaran data yang menggunakan format penulisan *JSON*. Salah satu keunggulan data *JSON* adalah ringan digunakan, dikarenakan penulisan format *JSON* yang sederhana. Sehingga ukurannya lebih kecil dibanding *XML*. Selain itu, keunggulan dari *JSON* yang lain adalah dapat digunakan dalam berbagai platform, sehingga dalam penggunaanya *JSON* merupakan format pertukaran data yang efisien. Format pertukaran data *JSON* sangat banyak digunakan untuk saat ini. Dikarenakan beberapa alasan yang telah penulis sebutkan di paragraf pertama. Mulai dari perusahaan besar seperti google yang menyediakan *API* dengan format *JSON* sampai perusahaan yang baru merintis atau bahkan perorangan/komunitas (The Official Buzz from Blogger at Google, 2017). Selain google yang termasuk salah satu perusahaan besar menyediakan dan memfasilitasi API berupa *JSON* data, pastinya *user* atau pengguna *JSON* data dari google API pun tidak sedikit pula. Berikut merupakan gambar penggunaan data *JSON* yang semakin meningkat dan sudah mencapai hitungan yang banyak

Historical trends in the usage of structured data formats for websites

This report shows the historical trends in the usage of structured data since August 2017.

	2017 1 Aug	2017 1 Sep	2017 1 Oct	2017 1 Nov	2017 1 Dec	2018 1 Jan	2018 1 Feb	2018 1 Mar	2018 1 Apr	2018 1 May	2018 1 Jun	2018 1 Jul	2018 1 Aug	2018 3 Aug
None	57.6%	57.0%	56.5%	56.0%	55.6%	55.1%	54.8%	54.1%	53.4%	52.9%	52.5%	52.1%	51.5%	51.5%
RDFa	36.6%	37.1%	37.6%	38.0%	38.4%	38.9%	39.2%	39.8%	40.4%	40.9%	41.2%	41.6%	42.1%	42.2%
Meta-tag-based formats	18.0%	18.4%	18.8%	19.2%	19.7%	20.1%	20.5%	21.1%	21.6%	22.1%	22.5%	22.9%	23.4%	23.4%
JSON-LD	16.2%	16.6%	17.0%	17.3%	17.7%	18.1%	18.4%	18.9%	19.3%	19.7%	20.1%	20.4%	20.8%	20.8%
Microdata	12.2%	12.4%	12.6%	12.8%	13.0%	13.1%	13.3%	13.6%	13.9%	14.2%	14.5%	14.8%	15.1%	15.1%
Microformats	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%

Gambar 1.1 *Trends* penggunaan format data Internet (Gelbmann & Delamer, 2018).

OWL ontology dapat digunakan untuk membangun *semantic web* atau atau web semantik. Dalam membuat web semantik salah satu komponen terpenting adalah ontology/basis pengetahuan. Karena yang membedakan web semantik dan web biasa adalah adanya basis pengetahuan pada web semantik yang dapat membantu web untuk lebih memahami kebutuhan user. Salah satu bentuk ontology atau basis pengetahuan dapat berupa *OWL* yang digunakan oleh sistem bukan hanya menampilkan informasi namun juga memproses informasi data dan ditulis dengan format *RDFS* (Horrocks et al., 2004).

Salah satu yang mendasari pembuatan aplikasi konversi ini adalah pentingnya mempunyai basis pengetahuan untuk membangun sebuah web semantik. Karena dalam proses pembuatan web semantik akan lebih mudah saat sudah mempunyai *OWL* ontology (basis pengetahuan). Selain itu berikut adalah beberapa aplikasi yang berbasis web semantik. *Services Mashups The New Generation of Web Applications* (Benslimane, Dustdar, & Sheth, 2008), *Haystack: A Platform for Penulising End User Semantic Web Applications* (Quan, Huynh, & Karger, 2003), Piazza: Data Management Infrastructure forSemantic Web *Applications* (Halevy, Ives, Mork, & Tatarinov, 2003) dan masih banyak lainya.

Berikut adalah gambaran proses yang dilakukan dalam merancangn sebuah ontology:

- Tentukan domain dan ruang lingkup ontologi
- Pertimbangkan untuk menggunakan kembali ontologi yang ada
- Menghitung istilah penting dalam ontologi
- Mentukan kelas dan hierarki kelas
- Mentukan sifat kelas-slot
- Mentukan sisi slotnya
- Buat *instance* (Natalya F. Noy & Deborah L. McGuinness, 2000)

Seperti yang sudah penulis jelaskan di atas untuk membuat *OWL ontology* bukan hal yang mudah. Karena yang membedakan ontology dengan format data lain adalah keunggulan di mana ada relasi atau sebuah pengetahuan yang dapat dipakai oleh sistem dan itu tidak dimiliki format data lain. Beberapa sumber sudah menyediakan ontologi yang bisa langsung digunakan, tapi untuk menggunaan yang spesifik masih sangat kurang jika hanya mengandalkan ontologi yang sudah ada (Miličić, 2014). Oleh karena itu salah satu solusi yang penulis tawarkan adalah sebuah aplikasi konversi format data *JSON* ke dalam *OWL ontology*, penulis menggunakan *JSON* karena yang sedang popular saat ini dan dari semua format data *JSON* memiliki keunggulan dibanding format data lain.

1.2 Rumusan Masalah

Rumusan masalah berikut akan dijawab oleh penelitian ini:

a. Bagaimana *mapping* atau pemetaan konversi data *JSON* ke dalam *OWL ontology*.
b. Bagaimana model *system* konversi yang dapat menghasilkan *ontology* dengan tidak mengurangi informasi atau pengetahuan dari data aslinya?

1.3 Batasan Masalah

Untuk menjaga fokus penelitian dalam TA ini, beberapa batasan berikut diperhatikan:

a. Format data yang digunakan dalam konversi berupa *JSON* data yang tervalidasi sesuai *JSON schema* (Pezoa et al., 2015) .
b. Fokus dari TA ini adalah melakukan analisis bagaimana *mapping* dari *JSON* data ke *OWL ontology* dalam bahasa Indonesia baku.

1.4 Tujuan Penelitian

Berikut adalah tujuan utama dari penelitian yang dilakukan

a. Menghasilkan analisis *mapping* untuk melakukan konversi file *JSON* data ke dalam *OWL ontology*
b. Menghasilkan aplikasi yang akan mempermudah user dalam membuat sebuah basis pengetahuan berupa *OWL* ontology dengan memanfaatkan data yang sudah ada (*JSON* data).

1.5 Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat berupa:

a. Mempermudah *developer*, khususnya *developer semantic web* dalam menggembangkan sebuah web semantic
b. Mempermudah developer dalam membuat sebuah basis pengetahuan berupa *OWL ontology*

1.6 Metode Penelitian

Metode penelitian dilakukan agar proses pembuatan sistem dapat dilakukan sesuai dengan rencana dan mendapatkan hasil yang diharapkan. Adapun metodologi yang diterapkan dalam penelitian ini yaitu:

a. Analisis

Pada tahap ini dilakukan analisis untuk mengetahui kebutuhan yang diperlukan dalam pembuatan sistem, seperti identifikasi masalah, usulan solusi, kebutuhan data, dan kebutuhan sistem.

b. Perancangan

Pada tahap ini dibuat rancangan yang diperlukan untuk menjadi acuan dalampembuatan sistem agar sistem dapat berjalan sesuai dengan kebutuhan pengguna. Rancangan tersebut antara lain use case diagram, activity diagram, struktur dan relasi tabel basis data, serta perancangan antarmuka.

c. Implementasi

Pada tahap ini dilakukan penerjemahan dari rancangan yang telah dibuat pada tahap sebelumnya ke dalam bahasa pemrograman.

d. Pengujian

Pada tahap ini dilakukan pengujian terhadap sistem yang telah dikembangkan. Pengujian diperlukan untuk memeriksa atau melakukan perbaikan apabila terdapat kesalahan (*bugs*) atau tidak kesesuaian pada sistem.

1.7 Sistematika Penulisan

Sistematika penulisan ditujukan untuk memudahkan dalam melakukan pembahasan tugas akhir ini. Secara garis besar, sistematika penulisan laporan ini terbagi menjadi enam bab. Adapun uraian dari masing-masing bab tersebut yaitu:

Bab I Pendahuluan, berisi latar belakang masalah, batasan masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian, dan sistematika penulisan yang dijadikan gambaran dan materi mengenai penelitian yang dilakukan.

Bab II Landasan Teori, berisi teori-teori yang berkaitan dan menjadi dasar dalam penelitian. Adapun teori-teori tersebut yaitu yang berhubungan dengan data *JSON*, data *OWL, JSON schema,* konversi data *JSON* dan penelitian terdahulu.

Bab III Analisis dan Perancangan Sistem, berisi analisis identifikasi masalah, gambaran umum sistem, solusi penyelesaian masalah, dan analisis kebutuhan yang diperlukan dalam penelitian. Serta analisis sistem yang dibuat dalam bentuk diagram *UML* (*Unified Modelling Language*), struktur dan relasi *table* basis data serta rancangan antarmuka sistem.

Bab IV Implementasi dan Pengujian Sistem, berisi hasil implementasi dari rancangan yang telah dibuat, pengujian sistem dan hasil evaluasi calon pengguna terhadap sistem yang telah dikembangkan.

Bab V Kesimpulan dan Saran, berisi kesimpulan yang merupakan rangkuman dari keseluruhan hasil penelitian dan memberikan saran untuk perbaikan kekurangan yang ditemukan dalam penelitian ini untuk dikembangkan lebih lanjut.

BAB II
LANDASAN TEORI

2.1 *JSON*

JSON merupakan format pertukaran data. Dengan menggunakan API (*Application programming interface*) data dapat dipertukarkan atau digunakan dengan format penulisan *JSON*. *JSON* terbuat dari dua struktur, yaitu:

a. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (object), rekaman (record), struktur (struct), kamus (dictionary), tabel hash (hash table), daftar berkunci (keyed *list*), atau associative array (Bray, 2014).
b. Daftar nilai terurutkan (an ordered *list* of values). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (array), vektor (vector), daftar (*list*), atau urutan (sequence) (Bray, 2014).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini.

2.1.1 Alasan Menggunakan Data *JSON*

Berikut merupakan alasan penulis mengembngkan aplikasi konversi otomatis ini menggunakan data *JSON*:

a. *JSON* merupakan format pertukaran data yang penulisanya sederhana sehingga mudah dibaca dan ditulis oleh manusia
b. Karna penulisanya yang sederhana *JSON* juga memiliki keunggulan ringan dalam pertukaran data
c. Support untuk *JSON* sendiri sangat besar karena hampir kebanyakan pertukaran data sekarang menggunakan format *JSON* mulai dari perusahaan besar seperti *google* atau bahkan perusahaan kecil dan perorangan.
d. *JSON* sendiri mendukung berbagai *platform* sehingga dari segi penggunaan sangat efisien(Pezoa et al., 2015)

2.1.2 *JSON Schema*

Dalam penulisan data *JSON* membutuhkan sebuah *schema* penulisan di mana akan mengatur data *JSON* tersebut agar sesuai dengan standar dari *JSON schema.* Selain itu dalam penelitian ini penulis menggunakan *JSON schema* untuk melakukan analisis data *JSON* yang dimasukan. Sehingga penulis akan mendapatkan *properties* dari *JSON schema* yang sudah ada, kemudian dari hasil *parsing* akan dilakukan pemetakan untuk mendapatkan *list* yang dibutuhkan dalam membentuk file *OWL ontology.*

2.2 *OWL Ontology*

OWL ontology adalah salah satu basis pengetahuan yang dapat digunakan untuk membangun *semantic web* atau atau web semantik. Dalam membuat web semantik salah satu komponen terpenting adalah ontology/basis pengetahuan. Karena yang membedakan web semantik dan web biasa adalah adanya basis pengetahuan pada web semantik yang dapat membantu web untuk lebih memahami kebutuhan user.

Salah satu yang mendasari pembuatan aplikasi konversi ini adalah pentingnya mempunyai basis pengetahuan untuk membangun sebuah web semantik. Karena dalam proses pembuatan web semantik akan lebih mudah saat sudah mempunyai *OWL* ontology (basis pengetahuan).

Berikut adalah gambaran proses yang dilakukan dalam merancangn sebuah ontology:

a. Tentukan domain dan ruang lingkup ontology
b. Pertimbangkan untuk menggunakan kembali ontologi yang ada
c. Menghitung istilah penting dalam ontology
d. Mentukan kelas dan hierarki kelas
e. Mentukan sifat kelas-slot
f. Mentukan sisi slotnya
g. Buat *instance* (Natalya F. Noy & Deborah L. McGuinness, 2000)

2.3 Penelitian Terdahulu

Sistem konversi otomatis data *JSON* ke *OWL Ontology* setelah penulis dan pembimbing cari belum menemukan sistem yang serupa. Di mana memanfaatkan data *JSON* yang begitu banyak untuk mendapatkan data *OWL Ontology* yang secara struktur lebih kaya dari pada data *JSON* itu sendiri. Untuk itu juga salah satu alasan penulis dan pembimbing sepakat dalam mengembangkan sistem ini.

Untuk penelitian yang hampir serupa penulis menemukan sistem konversi data *XML* ke *OWL Ontology*. Dan juga pada disertasi pembimbing tentang penelitian Data *JSON* ke *OWL Ontology*. Pada kedua referensi penelitian terdahulu membahas tentang metode baru dalam membuat data *OWL Ontology* dengan cara yang lebih mudah. Karena menggunakan data dalam bentuk lain yang nantinya melakukan akan dilakukan konversi untuk mendapat data baru berupa *OWL Ontology*.

Selain itu kenapa penulis mengembangkan sistem konversi ini, karena penulis belum menemukan sebuah sistem atau implementasi dari penelitian terhadulu sampai berbentuk sistem yang dapat digunakan oleh pengguna. Kebanyakan dari penelitian terhulu baru membahas sampai mana konversi dan mapping datanya belum masuk ke dalam implementasi. Berikut adalah tabel 2.1 yang memuat penelitian terdahulu tentang sistem yang sedang penulis kembangkan.

Tabel 2.1 Penelitian terdahulu

Pengarang	Judul	Bahasan
Dhomas Hatta Fudholi	*Data - Driven Dynamic Common Ontology* (Fudholi, 2016)	Analisis beberapa model data (*XML*, *JSON*, *RDB*, *etc*) dengan *OWL Ontrology*
Nora Yahia(1), Sahar A. Mokhtar(2) and AbdelWahab Ahmed(3)	*Automatic Generation of OWL Ontology from XML Data Source*	Membahas bagaimana memperoleh data *OWL Ontology* dengan data *XML* yang ada
Hannes Bohring* and Soren Auer+	*Mapping XML to OWL Ontologies*	Pemetakan data XML ke dalam *OWL* Ontology

BAB III
ANALISIS

3.1 Analisis Masalah

Dalam mengembangkan web semantik *developer* menggunakan *ontology* sebagai basis pengetahuannya. Yang paling membedakan perancangan web semantik adalah adanya *ontology* dalam pengembanganya dari pada web yang tidak menggunakan *ontology*. Seperti yang sudah penulis jelaskan di bab sebelumnya untuk membuat *OWL ontology* bukan hal yang mudah. Karena yang membedakan ontology dengan format data lain adalah keunggulan di mana ada relasi atau sebuah pengetahuan yang dapat dipakai oleh sistem dan itu tidak dimiliki format data lain. Sampai saat ini ada beberapa kendala dalam membuat ontology, yaitu:

a. Sulitnya membuat *ontology* dari nol, karena *ontology* adalah basis pengetahuan dan membuatuhkan *effort* besar untuk membuat basis pengetahuan dari nol.
b. Salah satu cara mengatasi untuk tidak membuat *ontology* adalah menggunakan ulang atau me *re-use* sebuah *ontology* yang sudah ada. Namun pada kenyataanya *ontology* yang sudah ada masih sangat terbatas dan kebanyakan *domain* dari *ontology* yang sudah ada tersebut sulit untuk dicari kecocokanya dengan *ontology* yang di butuhkan, terutama untuk penelitian baru yang memang belum pernah sama sekali ada yang membuat *ontology* seperti *domain* yang akan digunakan.
c. Begitu banyaknya data *JSON* dalam penggunaanya juga dapat dijadikan alas an atau masalah kenapa tidak membuat sebuah analisis pemetakan untuk membuat data *JSON* tersebut menjadi sebuah data yang kaya dan mungkin lebih berguna (dalam lingkup *semantic web*) .

3.2 Usulan Pemecahan Masalah

Dengan permasalahan yang sedang dihadapi oleh *developer* web semantik dalam mengembangkan web berbasis semantik. Disini penulis mencoba mengusulkan pemecahan masalah dengan mengembangkan sebuah sistem yang dapat menerima masukan data *JSON* kemudian dikonversi menjadi sebuah *OWL Ontology* . Sehingga dengan demikian akan lebih mudah bagi *developer* untuk mengembangkan web semantik yang mereka rancang.

Sebuah aplikasi yang mampu membaca data *JSON* sehingga dapat melakukan analisis pada setiap *properties* data tersebut. Kemudian melakukan *mapping* / pemetakan dari hasil analisis dan *parsing* data *JSON*, sehingga didapatkan sebuah *list* objek dan *properties* lainya. Dan kemudian melakukan penyusukan file baru dengan format *OWL ontology*. Sehingga didapatkan sebuah basis pengetahuan yang dapat dipakai dalam dasar pengembangan web semantik yang akan dibuat.

3.3 Analisis Kebutuhan Informasi

Analisis kebutuhan informasi data dilakukan untuk mengetahui informasi – informasi penting dari data *JSON* yang akan dikonversi sehingga sistem tahu data mana yang akan dilakukan pemetakan. Setiap informasi data dapat dilihat dari *JSON schema* yang dilakukan parsing dan pengolahan datanya ke dalam bentuk *list* yang sudah dipisahkan berdasarkan beberapa karakter yang telah penulis tentukan sehingga data bisa langung digunakan untuk membentuk data *OWL ontology*. Berukut merupakan rincian kebutuhan informasi yang diperlukan oleh sistem:

a. Data *JSON*

   Value dari data *JSON* sangat dibutuhkan dalam pembantukan data *OWL ontology* baru nantinya.

b. Data *JSON schema*

   1. Objek dalam *JSON*
   2. *Type* data dari setiap *properties*
   3. Jumlah hirarki ke dalaman data *JSON*
   4. Setiap *properties* di semua hirarki
   5. Informasi kepemilikan anak disetiap hirarki

3.4 Analisis Kebutuhan Masukan

Analisis kebutuhan masukan dilakukan untuk mengetahui masukan-masukan apa saja yang diperlukan di dalam sistem. Kebutuhan masukan yang digunakan oleh pengguna di dalam sistem yaitu data masukkan berupa username dan password untuk login, data *JSON* untuk melakukan konversi ke dalam data *OWL ontology*

3.5 Analisis Kebutuhan Proses

Analisis kebutuhan proses merupakan analisis untuk mengetahui proses-proses apa saja yang diperlukan di dalam sistem. Berikut merupakan perincian kebutuhan proses yang terdapat di dalam sistem:

a. Proses *login*

   Sistem ini tidak hanya digunakan oleh satu pengguna. Sehingga dibutuhkan proses login untuk mengenali siapa yang akan menggunakan sistem dan hal apa saja yang dapat dilakukan pengguna tersebut di dalam sistem. Proses login ini dibutuhkan saat pengguna ingin menggunakan fitur bookmark, sehingga pengguna dapat menyimpan hasil konversi yang telah dilakukan sebelumnya.

b. Proses konversi data *JSON* ke *OWL ontology*

   Sistem akan melakukan banyak proses, mulai dari parsing, mapping, dan semua proses yang dibutuhkan sedemikianrupa lalu menghasilan data *OWL ontology*

c. Proses menambah *bookmark* file hasil konversi

   Pengguna dapat menambahkan *bookmark* kepada file hasil konversi jika ingin ingin menuyimpannya. Dengan demikian pengguna dapat melihat kembali hasil konversi tanpa perlu melakukan konversi lagi

d. Proses melihat *bookmark* file hasil konversi

   Sistem akan menampung hasil konversi dan akan melakukan cek ulang untuk melihat kembali data yang sudah dilakukan konversi.

e. Proses melihat *profile*

   Sistem akan menampung detail *profile* berupa informasi akun dan data lainya.

3.6 Analisis Kebutuhan Keluaran

Analisis kebutuhan keluaran dilakukan untuk mengetahui keluaran apa saja yang diperlukan di dalam sistem. Berikut merupakan perincian kebutuhan keluaran dari sistem untuk konversi data *JSON* ke dalam data *OWL ontology:*

a. Data *OWL* ontology hasil dari konversi data *JSON*.

b. Detail history dari menggunaan sistem, data yang sudah terkonversi dan tersimpan sesuai akun *login*.

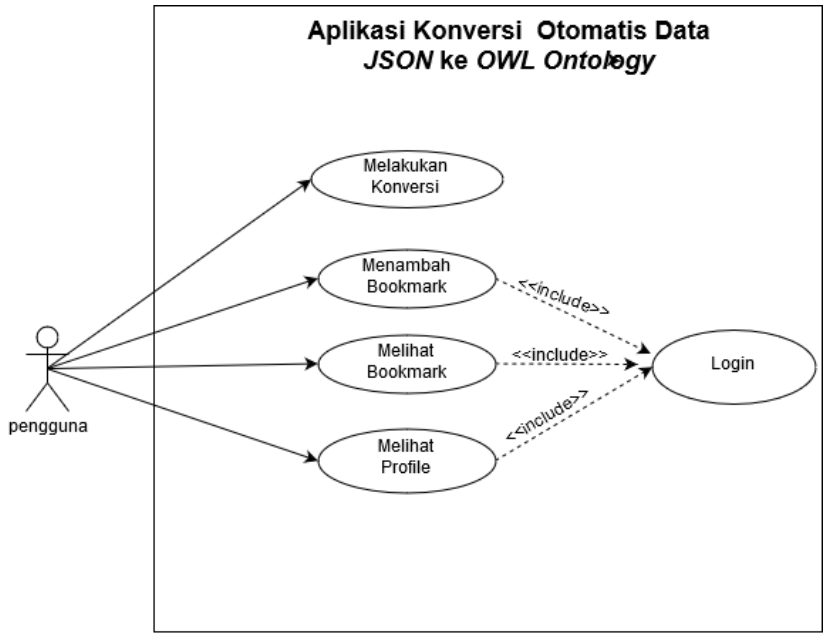
3.7 Analisis Kebutuhan Antarmuka

Dilakukan untuk mengetahui kebutuhan antarmuka dari sistem yang akan dibangun. Berikut merupakan perincian kebutuhan antarmuka di dalam sistem:

a. Antarmuka halaman *login* untuk melakukan autentikasi pengguna.
b. Antarmuka halaman *input* data *JSON* untuk memasukan data *JSON* dari *storate* sistem.
c. Antarmuka halaman *OWL ontology* untuk menampilkan hasil konversi data.
d. Antarmuka halaman *bookmark user* untuk melihat data yang disimpan.

3.8 Use case Diagram

Use case diagram metupakan suatu model untuk memberikan gambaran sistem secara keseluruhan. Diagram ini menggambarkan beberapa atau semua actor dan interaksi-interaksi yang terjadi dengan system untuk mengenalkan system yang dibuat. Adanya *use case* ini dapat memberikan informasi mengenai fungsi-fungsi apa saja yang terdapat pada sistem yang dibuat. *Use case* diagram yang terlihat pada Gambar 3.1 merupakan rancangan untuk sistem yang akan dibuat.



Gambar 3.1 *Use Case Diagram*

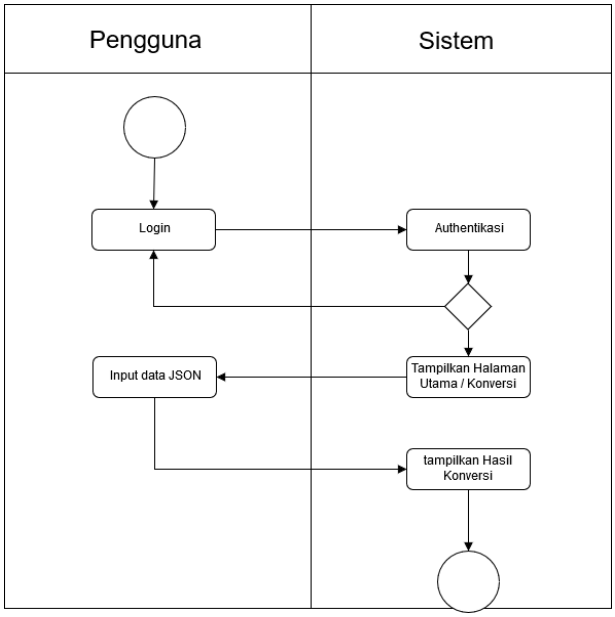
Dalam *use case* tersebut terlihat bahwa hanya terdapat satu aktor dalam sistem ini yaitu pengguna, yang berperan sebagai *developer*. Pengguna sistem dapat melakukan akses melakukan konversi, menambahkan *bookmark*, melihat *bookmark* dan melihat *profile*. Untuk dapat mengguanakan fungsi *bookmark* dan *profile* pengguna harus melakukan proses *login* terlebih dahulu. Berbeda dengan halaman *homepage* dimana pengguna tidak harus melakukan *login* terlebih dulu karena pada halaman ini pengguna dapat melakukan konversi data *JSON* ke *OWL Ontology*.

3.9 Activity Diagram

Activity diagram digunakan untuk menggambarkan aktivitas apa saja yang terjadi dalam sebuah sistem. Diagram ini menunjukan langkah-langkah dalam proses kerja sistem yang dibuat. Dalam sistem ini terdapat beberapa *activity diagram*, yaitu

3.9.1 *Activity* Diagram Melakukan Konversi

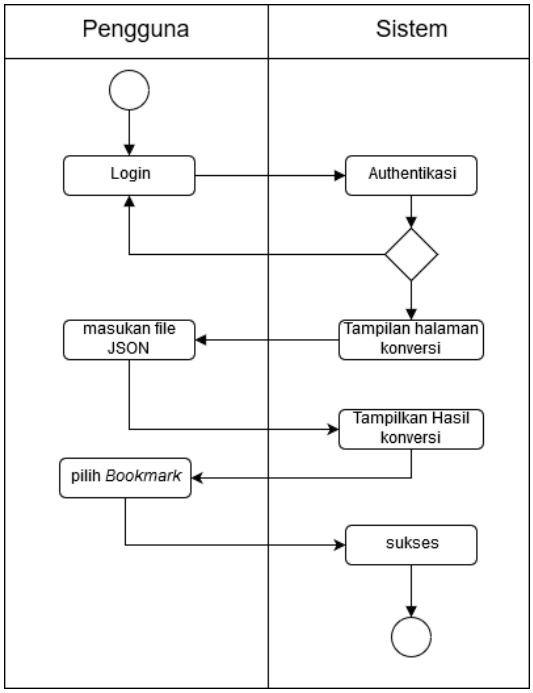
Diagram ini menggambarkan proses saat pengguna ingin menggunakan sistem untuk melakukan konversi data *JSON* ke *OWL Ontology*. Dalam proses ini pengguna memiliki 2 pilihan di mana dapat melakukan *login* terlebih dahulu atau tanpa perlu *login*. Setelah menentukan pilihan kemudian pengguna akan memasukan file *JSON* yang akan dikonversi. Selanjutnya sistem akan melakukan proses konversi untuk mengubah data *JSON* yang telah dimasukan. *Activity* diagram *homepage* bisa dilihat pada Gambar 3.2



Gambar 3.2 *Activity* diagram melakukan konversi

3.9.2 *Activity* Diagram Melakukan *Bookmark*

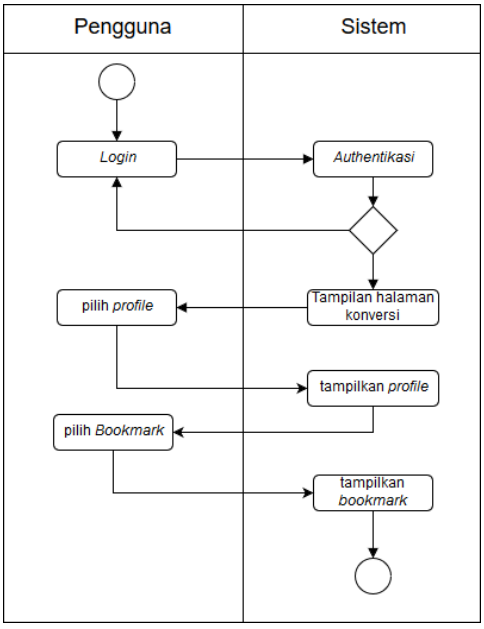
Diagram ini menggambarkan proses saat pengguna melakukan *bookmark* pada hasil konversi, dengan tujuan untuk menyimpan hasilnya kedalam *database.* Sehingga kalau ingin menggunakanya kembali tinggal melihat *list bookmark* tanpa perlu melakukan proses konversi lagi. Untuk *activity* diagram melakukan *bookmark* dapat dilihat pada gambar 3.3.



Gambar 3.3 *Activity* diagram melakukan *bookmark*

3.9.3 *Activity* Diagram Melihat *Bookmark*

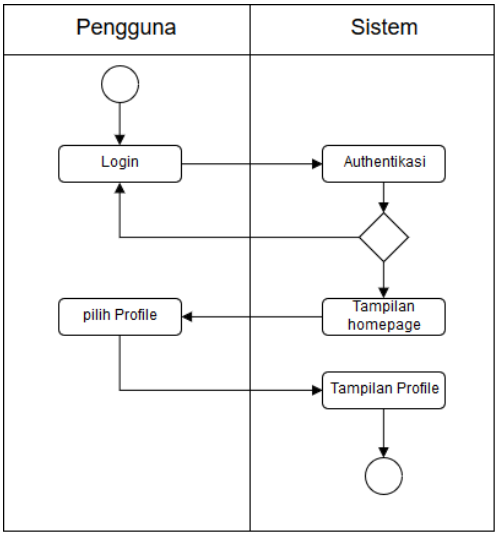
Diagram ini menggambarkan proses saat pengguna ingin menampilkan hasil bookmark dari data *JSON* yang sudah pernah dikonversi oleh sistem. Dalam proses ini pengguna harus *login* terlebih dahulu, karena data yang disimpan sesuai dengan akun yang digunakan untuk *login*. Untuk *activity* diagram melihat *bookmark* dapat dilihat pada gambar 3.4.



Gambar 3.4 Activity Diagram melihat Bookmark

3.9.4 Activity Diagram Melihat *Profile*

Diagram ini menggambarkan data dari pengguna sistem di mana sistem akan menampilkan data diri atau *profile* pengguna. Hal pertama yang silakukan tentu saja penguna harus *login* terlebih dahulu. Kemudian memilih menu *profile* dan sistem akan menampilkan data dari pengguna yang telah login. Berikut adalah diagram *avtivity* profile pada Gambar 3.5



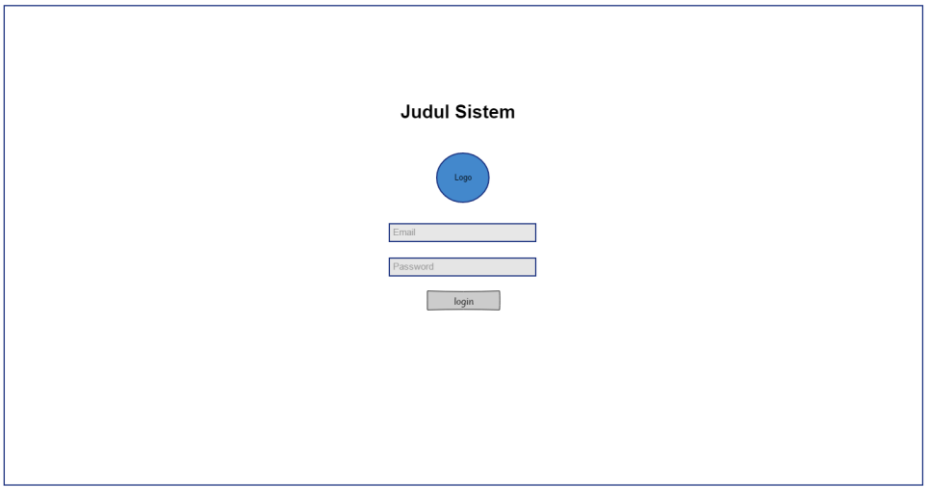
Gambar 3.5 Activity Diagram Profile

3.10 Rancangan Antar Muka

Rancangan antarmuka dibuat berdasarkan kebutuan antarmuka yang telah dianalisis sebelumnya. Rancangan ini dibutuhkan agar pembuatan sistem lebih bias mudah dilakukan dan terarah. Berikut merupakan rancangan antarmuka Aplikasi Konversi Otomatis Data *JSON* ke *OWL Ontology.*

3.10.1 Antarmuka Halaman *Login*

Rancangan antarmuka halaman *login* seperti yang terlihat pada gambar dibawah digunakan oleh pengguna untuk memasuk ke dalam sistem. Di halaman ini terdapat masukan berupa *email* dan *password*, serta sebuah tombol masuk untuk melakukan autentikasi masukan dari pengguna. Rancangan antar muka halaman *login* dapat dilihat pada Gambar 3.6



Gambar 3.6 Antarmuka halaman login

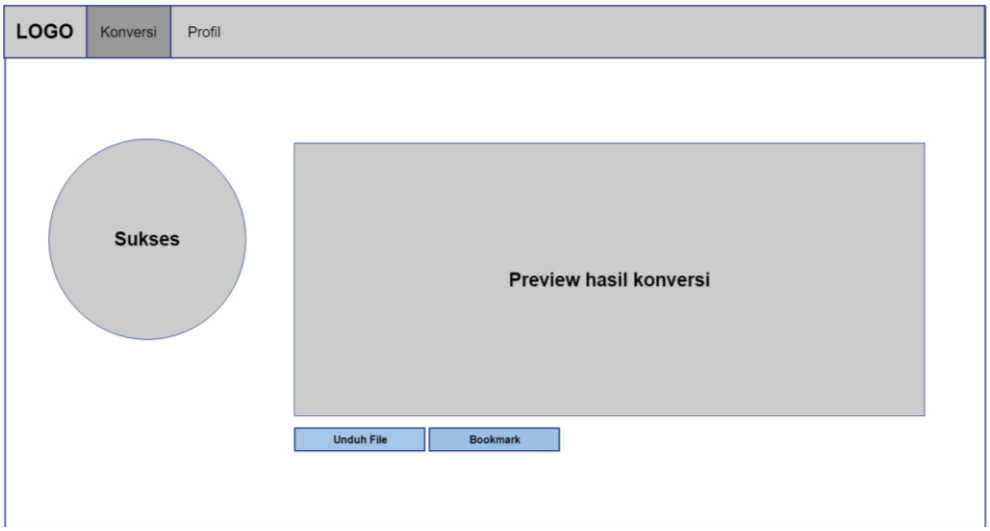
3.10.2 Antarmuka Halaman Konversi

Halaman ini adalah halaman utama dan pertama kali ditampilkan baik setelah pengguna *login* atau tanpa *login*. Karna yang membedakan pengguna *login* atau tidak adalah fitur *bookmark* dalam sistem ini. Jadi tanpa perlu login pengguna tetap bias melakukan konversi menggunakan sistem ini. Dalam halaman ini terdapat masukan berupa data *JSON* yang berguna untuk pengguna memasukan data *JSON* yang akan dikonversi. Selain itu ada satu tombol yang berguna menjalankan proses konversi setelah pengguna memasukan data *JSON*. Setelah 2 hal tersebut terpenuhi sistem akan melakukan proses untuk ke tahap selanjutnya. Berikut rancangan anatarmuka dapat dilihat pada Gambar 3.7.

Gambar 3.7 Antarmuka halaman Homepage

3.10.3 Antarmuka Halaman Hasil

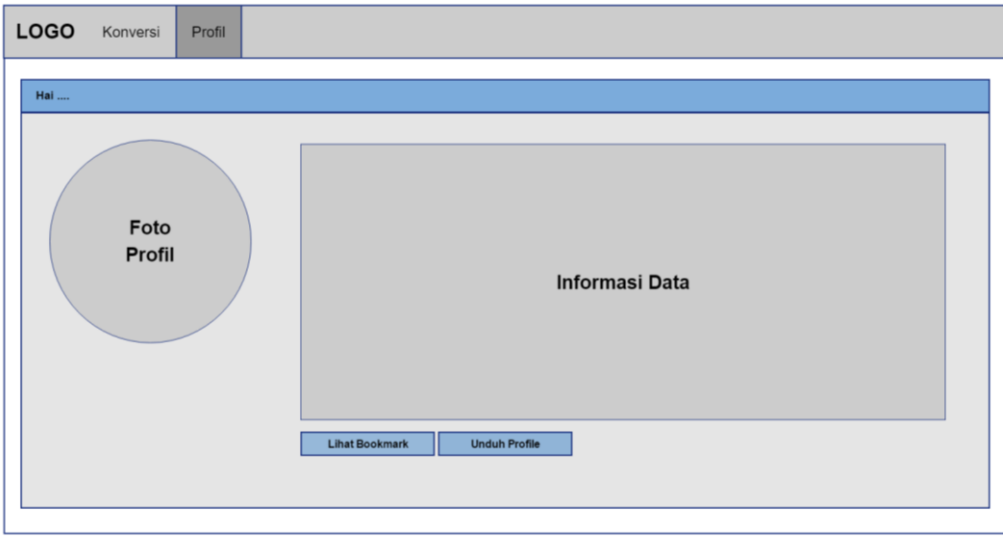
Setelah dari halaman konversi dan memasukan data *JSON* dan menekan tombol konversi sistem akan mengolah masukan dan menampilkan kehalaman selanjutnya. Seperti gambar dibawah ini adalah hasil dari konversi data *JSON* ke dalam *OWL Ontology*. Pada halaman ini menampilkan status dari konversi. Jika statusnya sukses akan menampilkan *preview* dari data yang sudah berhasil dikonversi. Kemudian ada 2 tombol untuk mengunduh hasilnya dan menyimpan ke dalam *bookmark*. Dapat dilihat pada Gambar 3.8



Gambar 3.8 Antarmuka halaman hasil

3.10.4 Antarmuka Halaman *Profile*

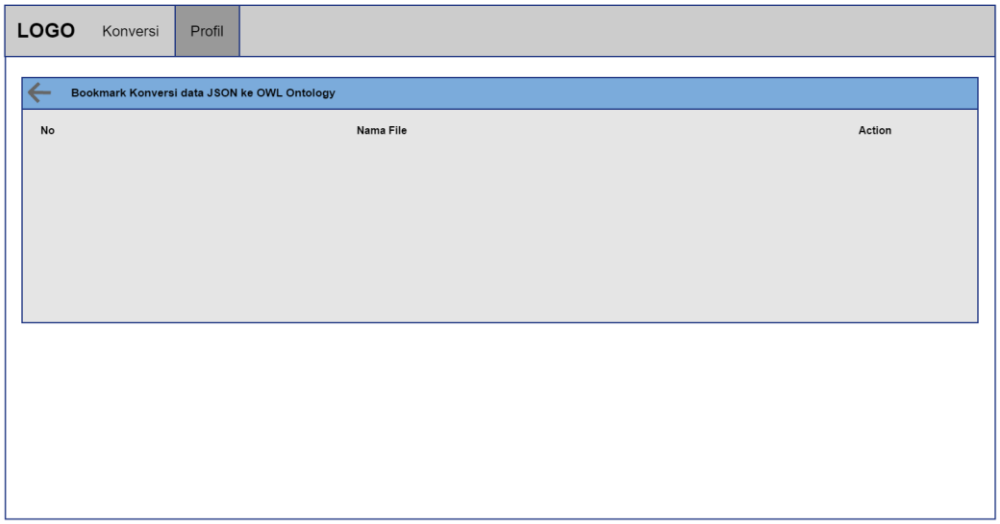
Dalam sistem yang penulis buat menyertakan *profile*. Didalam *profile* sendiri terdapat biodata dari pengguna. Selain itu dalam halaman *profile* juga dapat mengakses halaman *bookmark*. Di mana halaman *bookmark* adalah halaman yang menyimpan data hasil konversi sesuai akun dari pengguna. Dapat dilihat pada Gambar 3.9



Gambar 3.9 Antarmuka halaman profile

3.10.5 Antarmuka Halaman *Bookmark*

Antarmuka terakhir adalah halaman *bookmark*. Pada halaman ini menampilkan daftar hasil konversi yang telah disimpan atau ditandai oleh pengguna saat melakukan konversi adanya *bookmark* ini bertujuan supaya mempermudah penggguna dalam melakukan akses pada data yang telah dikonversi. Di mana data yang dikira penting dan memang perlu untuk ditandai atau disimpan akan masuk ke dalam daftar data konversi dihalaman *bookmark* ini. Rancangan antarmuka halaman *bookmark* dapat dilihat pada Gambar 3.10

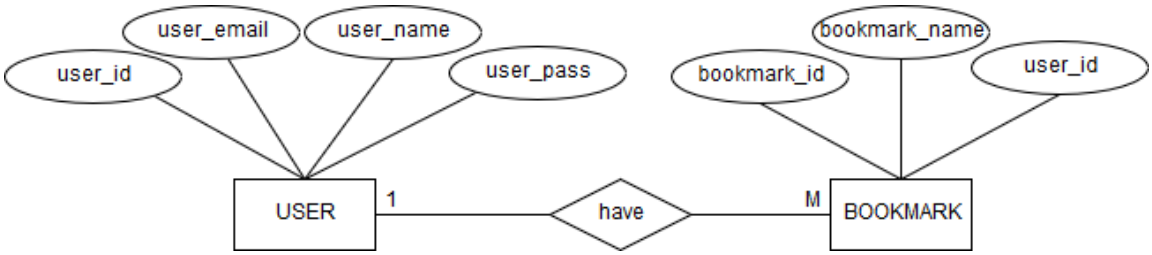


Gambar 3.10 Antarmuka halaman bookmark

3.11 Rancangan *ERD*

Dalam sistem yang penulis kembangkan membutuhkan sedikit *database* saja untuk menyimpan *bookmark.* Karena memang fokus utama dari sistem ini adalah untuk melakukan konversi sehingga dalam database masih tergolong sederhana.

Disini penulis membuat database dengan 2 entitas yang mana dihubungkan dengan relasi one to many (1:m). Sehingga setiap user dapat menyimpan lebih dari 1 data hasil konversi kedalam bookmark. Berikut adalah rancangan *ERD* yang penulis buat pada Gambar 3.11.



Gambar 3.11 Rancangan ERD

3.12 Memetakan data *JSON* ke *OWL Ontology*

Dalam melakukan mapping / pemetakan membutuhkan sebuah aturan. Disini menggunakan aturan "kesamaan karakter" dan "kesetaraan". Contoh:

Di dalam *ontology* ada *Class*, dan *class* itu mengumpulkan class lain di bawah nya atau bisa juga instance dari class tersebut. sedangkan didalam *JSON* memiliki *object* untuk membungkus *object* lain yang memiliki karakteristik serupa juga.

Dari aturan yang telah ditentukan untuk pemetakan data *JSON* ke *OWL Ontology* didapatkan table konversi yang dapat dilihat pada Tabel 3.1

Tabel 3.1 Konversi Atribut *JSON* ke *OWL Ontology*

No	Data *JSON*	*OWL* Ontology
1	*Object* yang memiliki karakteristik Umum	*OWL:Class*
2	*Property Object* yang bertipe *Object*	*OWL:objectProperty*
3	*Property Object* yang bertipe *Non Object*	*OWL:DatatypeProperty*
4	*Property Object* yang bertipe *Non Object* pertama	nama Individual
5	*Property Object* yang memiliki *value*	*Statement*

3.13 Modul dan Alur Pengerjaan Program

Dalam sistem yang sedang penulis kembangkan, penulis membagi pengerjaan sistem menjadi 4 modul. Di mana itu akan mempermudah proses pengerjaan karena penulis dapat fokus untuk melakukan pengembangan dan pengujuan kepada setiap modul yang telah dibagi. Selain itu pembagian ini juga bertujuan agar sistem lebih mudah dalam *maintenance* kedepanya karena diharapkan kedepanya sistem terus dikembangkan baik oleh penulis atau orang lain. berikut adalah pembagian modul dalam pengembangan sistem ini:

3.13.1 *Generate* data *JSON* ke dalam *JSON Schema*

Dalam modul pertama ini penulis membuat sebuah metode untuk menguraikan data *JSON* ke dalam *JSON Schema*. Penulis membutuhkan *JSON Schema* dikarenakan dalam melakukan konversi data *JSON* ke *OWL Ontology* yang dipakai untuk memetakan data adalah *JSON Schema* dari data yang dimasukan. Dalam modul ini penulis menggunakan bahasa *node.js.* Dikarenakan saat melakukan *research* penulis menemukan bahwa *node.js* memiliki dukungan *library* untuk modul ini.

3.13.2 Parsing dan Mapping data ke dalam sebuah *List* sesuai karakternya

Setelah mendapatkan *JSON Schema* dari modul sebelumnya kemudian akan dilakukan pengolahan pada modul ini. Di mana pada modul ini akan melakukan proses *parsing* dan mapping *JSON Schema* yang ada untuk kemudian dimasukan ke dalam setiap *list* yang telah dibuat. *List* disini bertujuan untuk klasifikasi level tiap hirarki dari file *JSON Schema* yang telah ada agar nantinya data yang diambil menjadi lebih banyak memiliki informasi untuk kemudian dapat dibentuk kembali menjadi file *OWL Ontology* pada modul selanjutnya.

Menurut penulis modul ini merupakan inti dari sistem yang dibuat di mana disini tantanganya adalah *file* yang dimasukan dalam sebuah program adalah *file* acak sehingga sistem dapat melakukan konversi otomatis terhadap semua data *JSON* yang dimasukan. Jadi pada modul ini algoritma yang digunakan penulis lebih rumit dan menentukan sistem yang dibuat apakah sesuai dengan tujuan awal atau tidak. Dalam modul ini penulis menggunakan bahasa pemrograman *java* dengan *library JSON simple.*

3.13.3 Menyusun menyusun *list* ke dalam *OWL* Ontology

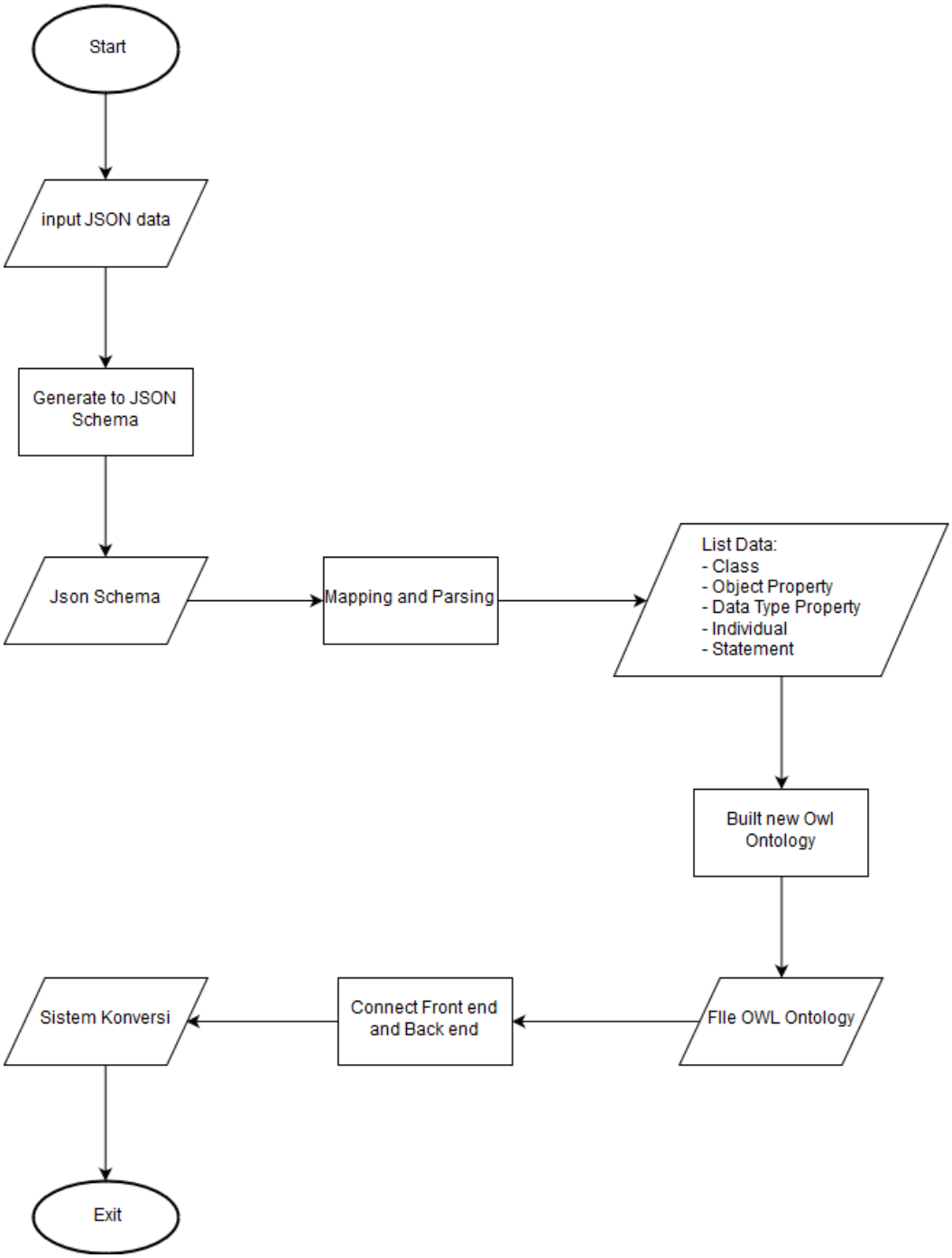
Setelah didapatkan *list* dari modul sebelumnya, pada modul ini penulis membuat program yang dapat menerima *list* dari modul sebelumnya kemudian dari *list* tersebut akan dibentuk kembali ke dalam sebuah data *OWL Ontology*. Dari hasil dari pembentukan file tersebut akan dibuat kelas untuk menjalankan setiap fungsi *generate file OWL Ontology*. Selanjutnya akan digunakan oleh modul terakhir / modul *interface* dalam menampilkan hasil akhir dari keseluruhan sistem yang dibuat. Dalam modul ini penulis masih mengguanakan bahasa pemrograman *java* namun dengan *library jena.*

3.13.4 Client dan *interface* sistem

Dalam modul terakhir ini berisikan program yang akan menampilkan hasil dari ketiga modul sebelumnya. Bisa dikategorikan untuk 3 modul sebelumnya adalah back-end dan modul ini adalah front-end. Di mana dalam modul inilah yang akan terlihat oleh pengguna. Dalam modul ini penulis menggunakan *html, css, java script dan jquery* untuk menampilkan data dan hasil dari pengolahan diketiga modul sebelumnya.

3.13.5 Flowchart Modul 1 sampai 4

Seperti yang sudah penulis jelaskan sebelumnya bahwa sistem ini dibagi ke dalam 4 modul pengerjaan. Setiap modul memiliki peran yang penting dalam memperoleh hasil konversi. Berikut adalah flowchart yang penulis rancang dapat dilihat pada Gambar 3.12



Gambar 3.12 Flowchart modul 1 sampai 4

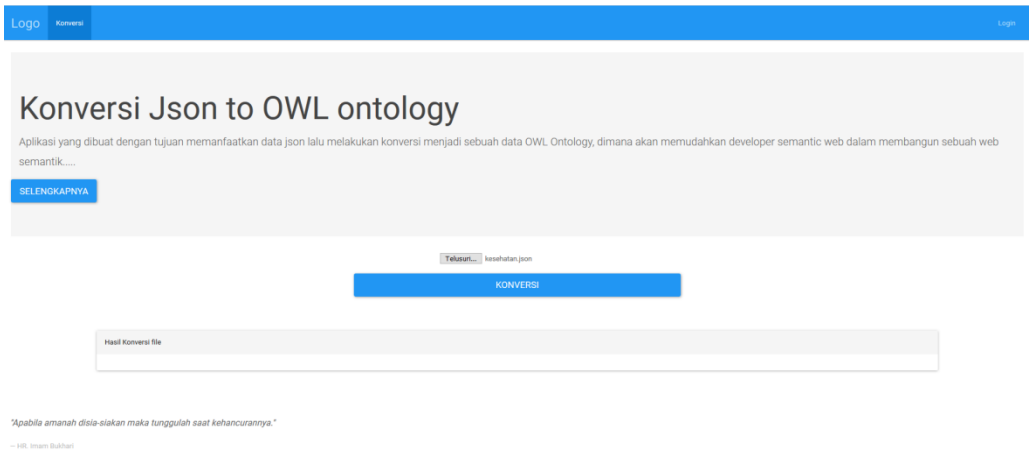
BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil

Hasil merupakan implementasi dari rancangan sistem yang telah dikerjakan pada tahap sebelumnya sehingga sistem siap untuk dioperasikan dan diuji kelayakannya dengan pengujian yang sudah ditetapkan.

4.1.1 Halaman Utama Sistem

Halaman utama atau home merupakan satu satunya halaman yang akan dijumpai oleh pengguna. Di mana pada halaman ini pengguna dapat menginputkan file *JSON* yang nantinya akan diolah oleh sistem sehingga mendapatkan file *OWL Ontology*. Hasil halaman utama dapat dilihat pada gambar 4.1.



Gambar 4.1 Tampilan aplikasi konversi Otomatis Data *JSON* ke *OWL Ontology*

4.1.2 File *OWL Ontology*

Setelah pengguna menggunakan sistem. Maka sistem akan mengolah sedemikian rupa sehingga sistem akan membuat *file* baru berupa *OWL Ontology*. Berikut adalah salah satu contoh perbandingan masukan dan keluaran

4.2 Pembahasan

Pada pembahasan penulis akan sedikit membahas bagaimana sistem bekerja secara teknis. Disini penulis membagi pengerjaan kadalam 4 bagian. Karena secara garis besar sistem yang sedang dikembangkan ini terdiri dari 4 modul utama.

4.2.1 Modul 1 - *Generate* file *JSON* ke *JSON Schema*

Pada modul pertama ini adalah permulaan dari alur sistem. Di mana yang pertama harus penulis kerjakan adalah melakukan konversi atau *generate* dari file *JSON* yang telah dimasukan untuk dijadikan *JSON Schema*. Pada sistem ini harus melalui proses ini karena penulis membuat sistem yang akan menguraikan / mengolah data dari suatu file. Untuk melakukan itu penulis harus tau informasi data / skema dari file yang akan diolah.

Pada modul ini penulis menggunakan bahasa pemrograman *Node.js.* Alasan penulis menggunakananya adalah karena *Node.js* merupakan salah satu bahasa dengan ketersediaan *library* yang terbilang sangat lengkap dan salah satunya adalah library untuk *generate* data *JSON* ke *JSON Schema*. Di mana dalam hal ini akan memudahkan penulis dalam mengerjakan modul ini.

Dalam modul ini penulis sedikit mengubah proses menjalankannya. Di mana modul ini penulis jalankan melalui bahasa *java*. Karena ke-2 modul lain dibuat menggunakan bahasa *java*. Maka untuk lebih mudah dalam intergrasi ke-4 modul dengan menggunakan *java* juga. Gambar 4.2 adalah potongan program yang menjadi jembataban antara modul ini dengan modul selanjutnya.

```
public Boolean GenerateJSONToSchema(String JSONFile, String JSONSchema)
throws IOException {

        String newfile = "-o";
        String replace = "-f";

        List<String> cmds = Arrays.asList("cmd.exe", "/C", "JSON-schema-
generator JSON-schema-generator\\" + JSONFile, "-f", "JSON-schema-
generator\\" + JSONSchema);
        ProcessBuilder builder = new ProcessBuilder(cmds);
        builder.directory(new
File("C:\\Users\\yaya_aye\\Documents\\NetBeansProjects\\Tugas_akhir_web")
);
        Process proc = builder.start();
        BufferedReader r;
        r = new BufferedReader(new
InputStreamReader(proc.getInputStream()));
        String line;
        String keluaran = null;
        while (true) {
            line = r.readLine();
            if (line == null) {
                break;
            }
            keluaran = keluaran + "\n" + line;
        }
        boolean check = new
File("C:\\Users\\yaya_aye\\Documents\\NetBeansProjects\\Tugas_akhir_web\\
JSON-schema-generator\\" + JSONSchema).exists();
        return check;
    }
```

Gambar 4.2 Modul 2 *generate JSON* ke *JSON* Schema

Singkatnya dalam potongan program di atas adalah kode program yang digunakan untuk menjalankan perintah yang akan menjalankan modul 1, dengan menggunakan 2 parameter. Namun pengguna hanya akan memasukan 1 masukan. Berupa file *JSON*. Sehingga modul ini akan menjalankan perintah untuk mengenerate file *JSON* ke dalam *JSON Schema* secara otomatis. Contoh hasil *generate file JSON* ke dalam *JSON Schema* yang didapat dari eksekusi modul 1 bisa dilihat pada Gambar 4.3 dan Gambar 4.4

```
{
  "kesehatan": {
    "penyakit": [
      {
        "namaP": "Malaria",
        "id": 101,
        "obat": {
          "namaO": "obatMalaria1",
          "kandungan": {
            "tumbuhan": "jahe",
            "senyawa": "Hcl"
          }
        }
      },
      {
        "namaP": "demam Berdarah",
        "id": 102,
        "obat": {
          "namaO": "obatDemamBerdarah",
          "kandungan": {
            "tumbuhan": "Temulawak",
            "senyawa": "NaC20"
          }
        }
      },
      { ··· },
      { ··· },
      { ··· },
      { ··· }
    ]
  }
}
```


Gambar 4.3 Contoh *file JSON* yang akan di *generate* ke *JSON Schema*

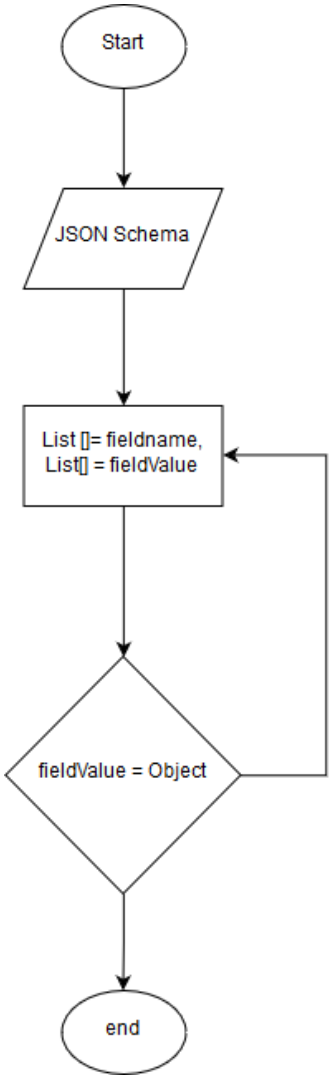
```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "description": "",
  "type": "object",
  "properties": {
    "data": {
      "type": "object",
      "properties": {
        "response": {
          "type": "object",
          "properties": {
            "count": {
              "type": "number"
            },
            "list": {
              "type": "object",
              "properties": {
                "kdDiag": {
                  "type": "string",
                  "minLength": 1
                },
                "nmDiag": { ··· },
                "nonSpesialis": { ··· }
              },
              "required": [ ··· ]
            },
            "list1": {
              "type": "object",
              "properties": { ··· },
              "required": [ ··· ]
            },
            "list2": { ··· }
          },
          "required": [ ··· ]
        },
        "metaData": { ··· },
        "kesehatan": { ··· }
      },
      "required": [ ··· ]
    }
  },
  "required": [
    "data"
  ]
}
```


Gambar 4.4 Hasil *Generate JSON* ke *JSON Schema*

4.2.2 Modul 2 – generate *JSON Scema* dan *JSON* ke dalam *List* Siap Pakai

Pada modul ini merupakan bagian program yang terpenting. Di mana salah satu implementasi dari penelitian ini adalah membuat sistem yang bisa melakukan pemrosesan data *JSON* menjadi *list* secara otomatis. Untuk kemudian hasil *generate*-nya dapat disusun menjadi file *OWL Ontology*.

Pada modul ini menggunakan 2 beberapa *library* yang sudah disediakan oleh *java*. Diantaranya adalah *library Jackson* dan *JSONPath*. Untuk *library Jackson* sendiri digunakan untuk melakukan proses *generate* data dari *JSON Schema* yang didapat dari memasukan data *JSON* dimodul pertama. Sedangkan untuk *library JSONPath* penulis gunakan untuk mencari data / *value* dari setiap *field* yang ada pada data *JSON* yang teah dimasukan sebelumnya. Karena berdasarkan pengalaman dan pencarian informasi oleh penulis belum menemukan cara *generate* semua data *JSON* beserta *field* dan keterangan dari setiap *field*-nya secara otomatis dan dinamis sesuai data *JSON* yang dimasukan. Oleh karna itu disini penulis melakukan generate *field* / property *JSON* terlebih dahulu dengan menggunakan *JSON Schema* sekaligus menentukan level hirarki pada setiap property dan menentukan *path* untuk setiap *field* yang mempunyai nilai/*value*. Setelah mendapatkan *path* dari propertinya penulis memasukan menggunakannya untuk mencari setiap nilai/*value* dengan menggunakan bantuan library *JSONPath*. Untuk lebih jelasnya penulis akan membahas dipoin selanjutnya. *Flowchart* yang telah penulis buat dapat dilihat pada Gambar 4.5 dan potongan program pada modul 2 ini pada Gambar 4.6.



Gambar 4.5 *Flowchart* Modul 2

```
public void TmpObjectAndProperty(JSONNode node, ArrayList<String>
listClass) {
            ArrayList<String> list = listClass;
            Iterator<String> fieldNames = node.fieldNames();
            while (fieldNames.hasNext()) {
                String fieldName = fieldNames.next();
                JSONNode fieldValue = node.get(fieldName);
                if (fieldValue.isObject() &&
!fieldName.equalsIgnoreCase("required")) {
                    if (!fieldName.equalsIgnoreCase("properties") &&
!fieldName.equalsIgnoreCase("items")) {
                        list.add(fieldName);
                    }
                    TmpObjectAndProperty(fieldValue, listClass);
                }
            }
        }
```

Gambar 4.6 Potongan kode program modul 2

Potongan kode program di atas adalah salah satu method yang penulis gunakan untuk melakukan *generate JSON Schema* sehingga mendapatkan *list* yang kemudian akan diolah lagi untuk mendapatkan property dari data *JSON* seperti *fieldname, value, path, hirarki* dan *list* yang akan disusun menjadi masukan untuk modul selanjutnya. Berikut adalah *list* yang dihasilan oleh kode program di atas. Di mana *list* ini didapat dari *generate JSON Schema* yang didapat dari keluaran modul 1. Untuk *JSON Schema* yang didapat dari modul 1 dapat dilihat pada Gambar 4.7 dan *list* yang hasil eksekusi kode program di atas dapat dilihat pada Gambar 4.8.

```
"$schema": "http://json-schema.org/draft-04/schema#",
"description": "",
"type": "object",
"properties": {
  "kesehatan": {
    "type": "object",
    "properties": {
      "penyakit": {
        "type": "array",
        "uniqueItems": true,
        "minItems": 1,
        "items": {
          "required": [
            "idP",
            "namaP"

          ],
          "properties": {
            "idP": {
              "type": "number"
            },
              "namaP": {
              "type": "string",
              "minLength": 1
            },
            "obat": {
              "type": "object",
              "properties": {
                "namaO": {
                  "type": "string",
                  "minLength": 1
                },
                "kandungan": {
                  "type": "object",
                  "properties": {
                    "tumbuhan": {
                      "type": "string",
                      "minLength": 1
                    },
                    "senyawa": {
                      "type": "string",
                      "minLength": 1
```

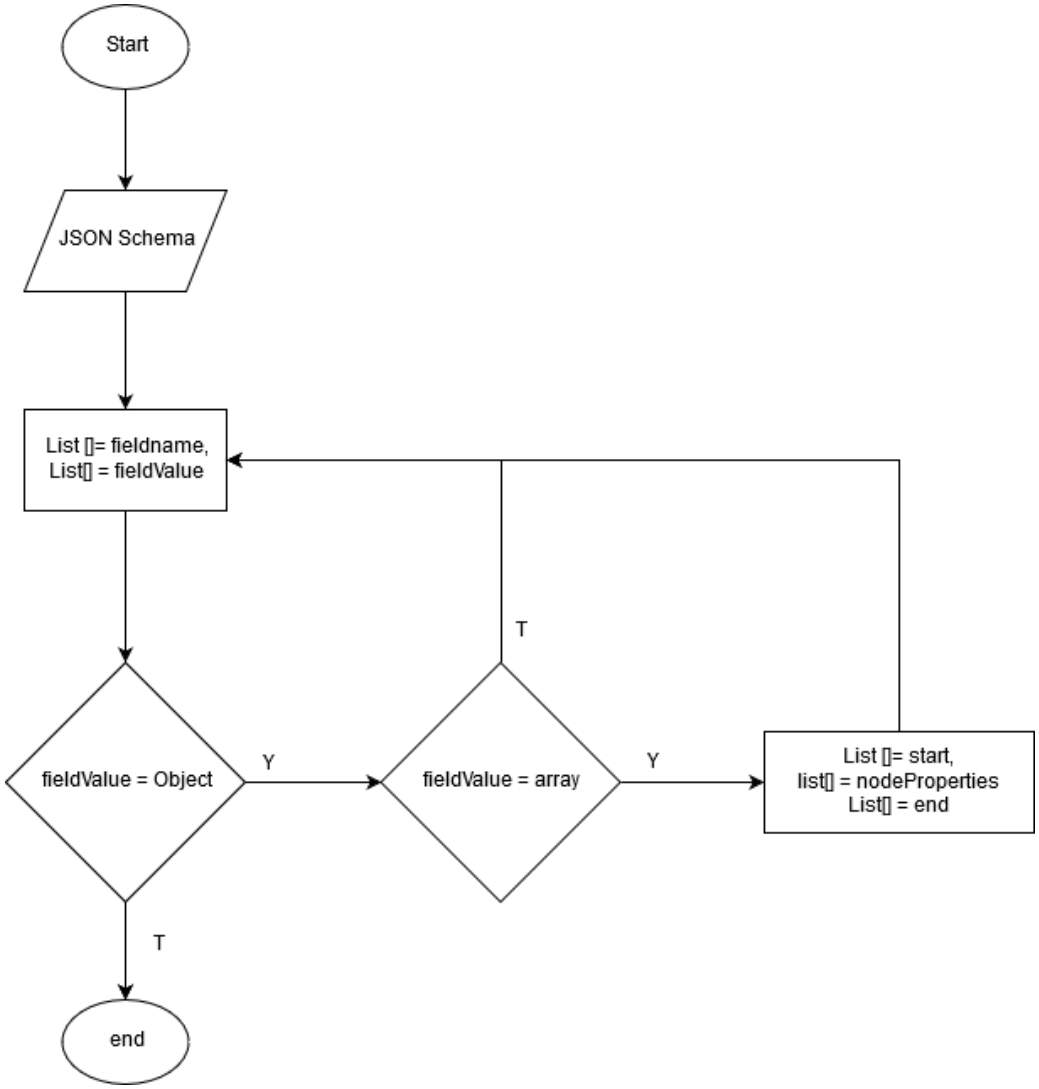
Gambar 4.7 *JSON Schema* masukan modul 2

```
Tmp objek and property:
[kesehatan, penyakit, idP, namaP, obat, namaO, kandungan, tumbuhan, senyawa]

[kesehatan, , type, object, properties, , penyakit, , type, array, uniqueItems, true, minItems, 1, requir
ed, , properties, , idP, , type, number, namaP, , type, string, minLength, 1, obat, , type, object, prope
rties, , namaO, , type, string, minLength, 1, kandungan, , type, object, properties, , tumbuhan, , type,
string, minLength, 1, senyawa, , type, string, minLength, 1, required, , required, , required, ]


list with information properties:
    [kesehatan, object]
    [penyakit, array]
    [idP, number]
    [namaP, string]
    [obat, object]
    [namaO, string]
    [kandungan, object]
    [tumbuhan, string]
    [senyawa, string]
```

Gambar 4.8 *List* temporari objek dan properti

Setelah mendapatkan 2 *list* yang memuat data yang masih belum bisa dibaca di atas, kemudian penuis membandingkan datany dan membentuk *list* baru dari erbandingan 2 *list* sebelumnya. Di mana didapat *list* ke 3 berupa nama *field*/property dari data *JSON* dan tipe atau jenis dari property tersebut. Apakah dia objek atau tipe data. Lalu dari *list* di atas sementara disimpan terlebih dahulu.

Kemudian penulis membuat *method* lagi untuk mencari beberapa *list* baru yang diperlukan untuk mendapat data yang bisa dibandingkan agar mendapatkan hasil atau data yang diinginkan. *Flowchart* yang penulis buat dapat dilihat pada Gambar 4.9 kemudian untuk *method GetAllElemen* dan *GetArray* dapat dilihat pada Gambar 4.10 dan Gambar 4.11



Gambar 4. 9 *Flowchart* untuk mendapatkan temporari objek dan properti

```
public void GetAllElemen(JSONNode node, ArrayList<String> listClass) {
        ArrayList<String> list = listClass;
        Iterator<String> fieldNames = node.fieldNames();
        while (fieldNames.hasNext()) {
            String fieldName = fieldNames.next();
            JSONNode fieldValue = node.get(fieldName);
            if (fieldValue.isObject()) {
                GetArray(fieldValue, listClass);
                list.add(fieldName);
                GetAllElemen(fieldValue, listClass);
            }
        }
    }
```

Gambar 4.10 *Method GetAllElemen*

```
public void GetArray(JSONNode node, ArrayList<String> listClass) {
        ArrayList<String> list = listClass;
        Iterator<String> fieldNames = node.fieldNames();
        while (fieldNames.hasNext()) {
            String fieldName = fieldNames.next();
            JSONNode fieldValue = node.get(fieldName);
            if (fieldValue.isArray()) {
                list.add("start");
                for (final JSONNode objNode : fieldValue) {
                    list.add(objNode.asText());
                }
                list.add("end");
            }
        }
    }
```

Gambar 4.11 *Method GetArray*

Dari potongan kode program di atas akan membuat 1 *list* yang mendefinisikan hirarki atau properti untuk setiap objek / *field*-nya. Di mana untuk setiap objek dan property akan dimasukan ke dalam 1 *list* dan sebagai penanda penulis menyisipkan beberapa *pointer* untuk mempermudah penulis mengolah data keproses selanjutnya. Di sini penulis menyisipkan penanda "*start*" dan "*end*" sebagai penanda awal dan akhir dari setiap properti untuk setiap *field* / *parent* dari propertinya. *List* yang nantinya akan penulis bandingkan untuk mendapatkan *list* baru yang lebih spesifik dapat dilihat pada Gambar 4.12.

```
Elemen:
[kesehatan, properties, penyakit, properties, idP, namaP, obat, properties, namaO, kandungan, properties, tumbuhan
, senyawa]

Elemen with array:
[start, penyakit, end, kesehatan, properties, penyakit, start, idP, namaP, end, items, properties, idP, namaP, sta
rt, namaO, kandungan, end, obat, properties, namaO, start, tumbuhan, senyawa, end, kandungan, properties, tumbuhan
, senyawa]
```

Gambar 4.12 *List GetElemen* dan *GetArray*

Seletah mendapatkan *list* di atas, penulis kembali membandingkan untuk mendapatkan *list* baru dengan nama "*tmp Class and property*". *List* baru ini akan menyimpan nama *parent* dan *child* dan dimasukan ke dalam 1 *list*. Namun ini masih dalam *list temporary*, di mana *parent* dan *child* disini masih tercampur antara *object property* dan *datatype property*. jadi masih akan ada serangkaian seleksi / pembandingan data lagi agar mendapatkan *list* siap pakai untuk dimasukan ke dalam modul selanjutnya. Kode program untuk melakukan perbandingan *list* di atas dapat dilihat pada Gambar 4.13.

```
ArrayList<String> tmpClassNProperty = new ArrayList();
if (item != null) {
            for (int i = 1; i < elementWithArrayFix.size(); i++) {
                if
(elementWithArrayFix.get(i).equalsIgnoreCase("properties") &&
!elementWithArrayFix.get(i - 1).equalsIgnoreCase("end")) {
                    String superClass = elementWithArrayFix.get(i - 1);
                    for (int j = i - 2; j >= 0; j--) {
                        if
(elementWithArrayFix.get(j).equalsIgnoreCase("start")) {
                            for (int k = j+1; k <
elementWithArrayFix.size(); k++) {
                                if
(elementWithArrayFix.get(k).equalsIgnoreCase("end")) {
                                    break;
                                }
                                tmpClassNProperty.add(superClass);

tmpClassNProperty.add(elementWithArrayFix.get(k));
                            }
                            break;
                        }
                    }
                }
            }
        }
```

Gambar 4.13 Kode program untuk mendapatkan list TmpClassAndProperty

Dari potongan kode program di atas akan menghasilkan *list* baru seperti yang sudah penulis jelaskan. Di mana dalam melakukan seleksi kali ini penulis menggunakan pointer "*start*" dan "*end*" yang sudah dimasukan ke dalam *list* yang diolah dikode program sebelumnya. Setelah mengilah berdasarkan aturan yang dibuat. *List* baru yang menyimpan *parent* dan *child* tapi namun masih belum di bedakan sesuai tipenya (*object property* dan *datatype property*) dapat dilihat pada Gambar 4.14.

```
Tmp Class and property
[[kesehatan, penyakit, penyakit, idP, penyakit, namaP, penyakit, obat, obat, namaO, obat, kandungan, kandungan, tum
buhan, kandungan, senyawa]
```

Gambar 4.14 *ListTmpClassAndProperty*

Kemudian setalah mendapatkan *list* parent dan child yang masih tercampurkan. Penulis melakukan seleksi untuk memisahkan mana yang *list* object property dan mana yang *list* datatype propery. Untuk kode programnya dapat dilihat pada Gambar 4.15

```
ArrayList<String> tmpObjectProperty = new ArrayList();
        ArrayList<String> tmpDatatypeProperty = new ArrayList();

        int point = tmpClassNProperty.size() - 1;
        for (int i = tmpClassNProperty.size() - 1; i >= 0; i = i - 2) {
            String theSub = tmpClassNProperty.get(point);
            String theSuper = tmpClassNProperty.get(point - 1);
            for (int j = 0; j < listWithProperties.length; j++) {
                if (theSub.equalsIgnoreCase(listWithProperties[j].get(0)))
{if      (listWithProperties[j].get(1).equalsIgnoreCase("object")      ||
listWithProperties[j].get(1).equalsIgnoreCase("array")) {
                        tmpObjectProperty.add(theSuper);
                        tmpObjectProperty.add(theSub);
                    } else {
                        tmpDatatypeProperty.add(theSuper);
                        tmpDatatypeProperty.add(theSub);

tmpDatatypeProperty.add(listWithProperties[j].get(1));
                    }
                }
            }point = point - 2;}
```

Gambar 4.15 Kode program untuk *listTmpObjectProperty*

Dalam kode program di atas melakukan perbandingan di mana *list parent* dan *child* sebelumnya akan dilakukan pengecekan. Apakah dia memiliki tipe *object* atau selain *object*. Saat diketahui hubungan antara *parent* dan *child* adalah *object* untuk keduanya. Maka dia akan dimaukan ke dalam *list* baru *tmp object property*. Jika ada diantara *parent* dan *child* yang selain *object*, akan dimasukan ke dalam *list tmp datatype property*. Hasil dari seleksi kode program di atas terhadap *list parent* dan *child* yang dapat dilihat pada Gambar 4.16.

```
tmp Object Property
[obat, kandungan, penyakit, obat, kesehatan, penyakit]

tmp Datatype Property
[kandungan, senyawa, string, kandungan, tumbuhan, string, obat, namaO, string, penyakit, namaP, string, penyakit, idP, number]
```

Gambar 4.16 *List tmpObjectProperty* dan *tmpDatatypeproperty*

Dari beberapa *list temporary* yang sudah dibuat. Penulis akan memasukanya lagi ke dalam *list* baru yang mana *list* baru ini adalah *list* yang siap dipakai dan dikirim sebagai input kemodul selanjutnya untuk diolah menjadi *OWL Ontology*. Berikut adalah sebagian siap pakai yang berhasil dikumpulkan dan dibuat dari serangkaian proses yang telah penulis lakukan. *List* data yang akan menjadi masukan di modul selanjutnya dapat dilihat pada Gambar 4.17.

```
-----------------------------------------data siap pakai-----------

- List Class:
    [kesehatan, penyakit, obat, kandungan]

- Object Property:
    [kesehatan, punyapenyakit, penyakit]
    [penyakit, punyaobat, obat]
    [obat, punyakandungan, kandungan]

- DataType Property:
    [penyakit, idP, xsd:integer]
    [penyakit, namaP, xsd:string]
    [obat, namaO, xsd:string]
    [kandungan, tumbuhan, xsd:string]
    [kandungan, senyawa, xsd:string]
```

Gambar 4.17 *List* data siap pakai

Saat ini penulis sudah memiliki 3 *list* yang siap pakai. Namun untuk membuat *OWL Ontology* dengan data *JSON* penulis masih membutuhkan beberapa *list* lagi yang harus didapatkan dari masukan yang ada. Pada langkah selanjutnya penulis akan membuuat *list* baru yang memuat *path* / jalur /susunan hirarki dari awal menuju *datatype property* sehingga dapat melakukan penelususan nilai / value dengan library *JSONPath*. Potongan kode program dan penjelasanya dapat dilihat pada Gambar 4.18.

```
for (int i = 0; i < buildListDataTypeProperty.length; i++) {
            String propertyName = buildListDataTypeProperty[i].get(1);
            String parrent = buildListDataTypeProperty[i].get(0);
            tmpListPathProperty[i].add(propertyName);
            for (int j = buildListObjectProperty.length - 1; j >= 0; j--)
{               if
(parrent.equalsIgnoreCase(buildListObjectProperty[j].get(2))) {
tmpListPathProperty[i].add(buildListObjectProperty[j].get(2));
                    for (int k = j; k >= 0; k--) {
                        if (k > 0) {
                            if (!buildListObjectProperty[k -
1].get(2).equalsIgnoreCase(buildListObjectProperty[k].get(0))) {

tmpListPathProperty[i].add(buildListObjectProperty[k].get(0));
                                break;
                            }
tmpListPathProperty[i].add(buildListObjectProperty[k - 1].get(2));
                        }
                    }
                }}}
```

Gambar 4.18 Kode program untuk mendapatkan pathDataTypeProperty

Dari kode program di atas penulis melakukan pengecekan untuk setiap *datatype property path* nya atau *parent* nya sampai ke *root* hirarki dari file *JSON*-nya. Ada beberapa kali memasukan ke dalam *list* terlebih dahulu sebelum disusun kembali menjadi *list* data yang siap pakai. Karena untuk data bertipe *Array*. Penulisan *root* nya berbeda. Jadi disini penulis menggunakan 2 *list* dan beberapa kali pengecekan. Hasil dari pengecekan dan penyusunan *path* mengambil nilai/*value* dari setiap *datatpe property* dapat dilihat pada Gambar 4.19.

```
tmp path property:
    [idP, penyakit, kesehatan]
    [namaP, penyakit, kesehatan]
    [namaO, obat, penyakit, kesehatan]
    [tumbuhan, kandungan, obat, penyakit, kesehatan]
    [senyawa, kandungan, obat, penyakit, kesehatan]

- Path class property:
    $.kesehatan.penyakit[*]idP
    $.kesehatan.penyakit[*]namaP
    $.kesehatan.penyakit[*]obat.namaO
    $.kesehatan.penyakit[*]obat.kandungan.tumbuhan
    $.kesehatan.penyakit[*]obat.kandungan.senyawa
```

Gambar 4.19 *List Path data type property*

Setelah mendapatkan *path* untuk mengakses *value*/nilai dari setiap *datatype property*. penulis membuat *method* dengan bantuan dari *library JSONPath* untuk mendapatkan semua nilainya. *Method* dan hasil dari *value* yang telah didapat sebelumnya bisa dilihat pada Gambar 4.20 dan hasilnya pada Gambar 4.21.

```
public void GetValueByPath(String fileName, String path,
ArrayList<String>[] list,int index ) throws FileNotFoundException,
IOException{
        String JSONFile = FileUtils.readFileToString(new
File("file/"+fileName), "UTF-8");
        list[index] = JSONPath.read(JSONFile, path);
}
   ArrayList<String>[] valueProperty = new
ArrayList[sizeListPathProperty];
        for (int i = 0; i < valueProperty.length; i++) {
            valueProperty[i] = new ArrayList<>();
        }
        for (int i = 0; i < buildListPathProperty.length; i++) {
            newGenerate.GetValueByPath(JSONFile,
buildListPathProperty[i].get(0), valueProperty,i);
        }
```

Gambar 4.20 *Method GetValueByPath*

```
value per property:
    [101,102,103,104,105,106,"idP"]
    ["Malaria","demam Berdarah","Diare","TBC","Tifus","Campak","namaP"]
    ["obatMalaria","obatDemamBerdarah","obatDiare","obatTBC","obatTifus","obatCampak","namaO"]
    ["jahe","Temulawak","DaunTeh","Kunyit","DaunPepaya","DaunUbiJalar","tumbuhan"]
    ["Hcl","NaC20","Np2H","H3P2","NaP4","k3P2","senyawa"]
```

Gambar 4.21 *List value data type property*

Setelah mendapatkan *value* untuk setiap *datatype property* kemudian penulis membutuhkan *list* baru yang menyimpan nama *individual* yang nantinya akan dimasukan ke dalam *OWL Ontology*. *List* individual ini didapat dari ***value datatype property* pertama dari suatu *object JSON*-nya.** Jadi disini penulis melakukan pengecekan lagi untuk nantinya menentukan dari setiap *datatype property* mana yang merupakan urutan pertama di-*class* tersebut.Potongan kode program untuk mendapatkan *list* individual dapat dilihat pada Gambar 4.22 dan hasilnya pada Gambar 4.23.

```
    ArrayList<String> tmpIndividual = new ArrayList<>();
    int counter;
         for (int i = 0; i < buildListDataTypeProperty.length; i++) {
             String clasIndividual = buildListDataTypeProperty[i].get(0);
             for (int j = i; j < buildListDataTypeProperty.length; j++) {
                 if ((j<buildListDataTypeProperty.length-1) &&
!buildListDataTypeProperty[j].get(0).equalsIgnoreCase(buildListDataTypePro
perty[j+1].get(0))) {

tmpIndividual.add(buildListDataTypeProperty[i].get(1));

tmpIndividual.add(buildListDataTypeProperty[i].get(0));
                     i=j;
                     break;
                 }else if(j==buildListDataTypeProperty.length-1){

tmpIndividual.add(buildListDataTypeProperty[i].get(1));

tmpIndividual.add(buildListDataTypeProperty[i].get(0));
                     i = j;
                     break;
                 }

             }

         }
```

Gambar 4.22 Kode program untuk mendapatkan *list* tmpIndividual

```
- List Individu name
    [101, penyakit]
    [102, penyakit]
    [103, penyakit]
    [104, penyakit]
    [105, penyakit]
    [106, penyakit]
    [obatMalaria, obat]
    [obatDemamBerdarah, obat]
    [obatDiare, obat]
    [obatTBC, obat]
    [obatTifus, obat]
    [obatCampak, obat]
    [jahe, kandungan]
    [Temulawak, kandungan]
    [DaunTeh, kandungan]
    [Kunyit, kandungan]
    [DaunPepaya, kandungan]
    [DaunUbiJalar, kandungan]
```

Gambar 4.23 *List individual*

Setelah megetahui *list* nama individu dan objeknya, penulis membutuhkan *list statement datatype property*. yaitu *list* yang berisi subyek, predikat dan objek. *List* ini adalah semacam *instance* dari individu yang telah dibuat. Di mana *list* ini di ambil dari *value datatype property*. Potongan kode program dapat dilihat pada Gambar 4.24 dan hasilnya pada Gambar 4.25.

```
    int pointerListStatement=0;
         for (int i = 0; i < buildListIndividual.length; i++) {
             String individuName = buildListIndividual[i].get(0);
             String className = buildListIndividual[i].get(1);
             for (int j = 0; j < buildListDataTypeProperty.length; j++) {
                 String classNameFromProperty =
buildListDataTypeProperty[j].get(0);
                 if(className.equalsIgnoreCase(classNameFromProperty)){
                     String propertyFromClassToProperty =
buildListDataTypeProperty[j].get(1);
                     int pointerLiteral=0;
                     for (int k = 0; k < valueProperty.length; k++) {
                         String propertyFromValue =
valueProperty[k].get(valueProperty[k].size()-1);
```

Gambar 4.24 Program untuk mendapatkan *list statement data type property*

```
- List Statemen:
    [101, punyaidP, 101]
    [101, punyanamaP, Malaria]
    [102, punyaidP, 102]
    [102, punyanamaP, demam Berdarah]
    [103, punyaidP, 103]
    [103, punyanamaP, Diare]
    [104, punyaidP, 104]
    [104, punyanamaP, TBC]
    [105, punyaidP, 105]
    [105, punyanamaP, Tifus]
    [106, punyaidP, 106]
    [106, punyanamaP, Campak]
    [obatMalaria, punyanamaO, obatMalaria]
    [obatDemamBerdarah, punyanamaO, obatDemamBerdarah]
    [obatDiare, punyanamaO, obatDiare]
    [obatTBC, punyanamaO, obatTBC]
    [obatTifus, punyanamaO, obatTifus]
    [obatCampak, punyanamaO, obatCampak]
    [jahe, punyatumbuhan, jahe]
    [jahe, punyasenyawa, Hcl]
    [Temulawak, punyatumbuhan, Temulawak]
    [Temulawak, punyasenyawa, NaC20]
    [DaunTeh, punyatumbuhan, DaunTeh]
    [DaunTeh, punyasenyawa, Np2H]
    [Kunyit, punyatumbuhan, Kunyit]
    [Kunyit, punyasenyawa, H3P2]
    [DaunPepaya, punyatumbuhan, DaunPepaya]
    [DaunPepaya, punyasenyawa, NaP4]
    [DaunUbiJalar, punyatumbuhan, DaunUbiJalar]
    [DaunUbiJalar, punyasenyawa, k3P2]
```

Gambar 4.25 *List statement data type property*

Langkah selanjutnya penulis akan melakukan *generate* untuk mendapatkan *list statement object property*, di mana *list* ini menyimpan subyek, predikat dan keterangan seperti *list statement data type property*, namun pada *list* ini diambil dari *object property* yang menjadi individual pada *list* sebelumnya. Potongan kode program dapat dilihat pada Gambar 4.26 dan hasilnya pada Gambar 4.27.

```
    ArrayList<String>[] buildListStatementOP= new
ArrayList[buildListIndividual.length - (valueProperty[0].size()-1)];
        for (int i = 0; i < buildListStatementOP.length; i++) {
            buildListStatementOP[i] = new ArrayList<>();
        }

        int pointerListStatementOP = valueProperty[0].size()-1;
        for (int i = 0; i < buildListStatementOP.length; i++) {
            String namaIndividu = buildListIndividual[i].get(0);
            String namaKelas =
buildListIndividual[pointerListStatementOP].get(1);
            String literal =
buildListIndividual[pointerListStatementOP].get(0);
            buildListStatementOP[i].add(namaIndividu);
            buildListStatementOP[i].add("punya"+namaKelas);
            buildListStatementOP[i].add(literal);
            pointerListStatementOP++;
        }
```

Gambar 4.26 Kode program untuk mendapatkan *list statement object poperty*

```
List Statement Object property:
    [101, punyaobat, obatMalaria]
    [102, punyaobat, obatDemamBerdarah]
    [103, punyaobat, obatDiare]
    [104, punyaobat, obatTBC]
    [105, punyaobat, obatTifus]
    [106, punyaobat, obatCampak]
    [obatMalaria, punyakandungan, jahe]
    [obatDemamBerdarah, punyakandungan, Temulawak]
    [obatDiare, punyakandungan, DaunTeh]
    [obatTBC, punyakandungan, Kunyit]
    [obatTifus, punyakandungan, DaunPepaya]
    [obatCampak, punyakandungan, DaunUbiJalar]
```

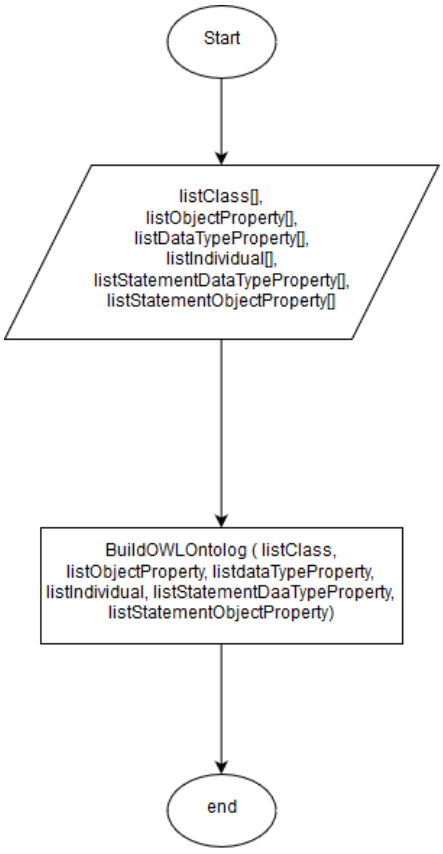
Gambar 4.27 *List statement object property*

Setelah beberapa langkah yang sudah penulis jelaskan di modul 2 untuk mendapatkan *list* yang menjadi masukan di modul selanjutnya. Penulis telah mendapatkan *list* yang sudah siap degenerate ke dalam *OWL Ontology*. Berikut adalah *list* dan hasil dari generate.

- *List class*
- *List object property*
- *List data type property*
- *List individual*
- *List Statement data type property*
- *List statement object property*

4.2.3 Modul 3 – Generate *List* Menjadi *OWL* Ontology

Pada modul ini penulis membuat kelas yang akan memuat *method* yang digunakan dalam melakukan *generate list* yang akan dimasukan dari modul sebelumnya menjadi *file OWL Ontology*. *Flowchart* yang penulis buat dapat dilihat pada Gambar 4.28 dan potongan kode programnya pada Gambar 4.29.



Gambar 4.28 *Flowchart* modul 3

```
public class BuildOntology {
    public void BuildOWLOntology(ArrayList<String> listClass,
ArrayList<String>[] listObjectProperty, ArrayList<String>[]
listDataTypeProperty, ArrayList<String>[] listIndividu,
ArrayList<String>[] listStatementDataTypeProperty, ArrayList<String>[]
listStatementObjectProperty) throws FileNotFoundException {
        String mergeURI = "http://www.example.com/merge.OWL#";
        PrintUtil.registerPrefix("merge", mergeURI);
        OntModel mo =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        mo.setNsPrefix("", mergeURI);
        mo.setNsPrefix("base", mergeURI);
```

Gambar 4.29 Potongan kode program modul 3

Dari potongan kode program di atas adalah awal kelas pada modul ini. Di mana penulis menggunakan *library jena* untuk melakukan generate *list* yang telah didapat dimodul sebelumnya. Setelah itu ada 1 method yang memuat beberapa *syntac* untuk memasukan *list*-nya. Namun pada potongan kode program di atas penulis lebih menekankan di mana method *BuildOWLOntoloy* memiliki 6 parameter yaitu *list* yang telah didapat dimodul 2. Kemudian untuk sebagian isi dari methodnya adalah untuk membuat model *ontology*. Jika dengan kode program di atas saja dijalankan maka hasilnya akan membentuk *file* yang memiliki prefix sesuai yang telah ditentukan di atas. Hasil dari potongan kode program pada Gambar 4.29 dapat dilihat pada Gambar 4.30.

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

Gambar 4.30 *File ontology prefiks NS*

Seletah itu penulis akan menjelaskan bagian bagian dari method dan *generate* setiap *list* yang dimasukan dari modul sebelumnya. Potongan kode program dari setiap langkah untuk generate *list*-nya dapat dilihat pada gambar 4.31 dan hasilnya pada gambar 4.32.

```
for (int i = 0; i < listClass.size(); i++) {
            mo.createClass(mergeURI + listClass.get(i));
 }
```

Gambar 4.31 Kode program membuat class

```
<owl:Class rdf:about="http://www.example.com/merge.owl#kandungan"/>
<owl:Class rdf:about="http://www.example.com/merge.owl#obat"/>
<owl:Class rdf:about="http://www.example.com/merge.owl#kesehatan"/>
<owl:Class rdf:about="http://www.example.com/merge.owl#penyakit"/>
```

Gambar 4.32 *File ontology* class

Kode program di atas akan melakukan perulangan sesuai isi dari *listClass* dan akan membuat atau mendefinisikan *class* didalam *OWL Ontology*. Setelah membuat *class* kemudian membuat *object property*. Potongan kode program dapat dilihat pada Gambar 4.33 dan hasilnya pada Gambar 4.34.

```
//         Create Object Property
        for (int i = 0; i < listObjectProperty.length; i++) {
            ObjectProperty oop = mo.createObjectProperty(mergeURI +
listObjectProperty[i].get(1));
            oop.addDomain(mo.getResource(mergeURI +
listObjectProperty[i].get(0)));
            oop.addRange(mo.getResource(mergeURI +
listObjectProperty[i].get(2)));
        }
```

Gambar 4.33 Kode program membuat *object property*

```
<owl:ObjectProperty
rdf:about="http://www.example.com/merge.owl#punyapenyakit">
  <rdfs:range rdf:resource="http://www.example.com/merge.owl#penyakit"/>
  <rdfs:domain
  rdf:resource="http://www.example.com/merge.owl#kesehatan"/>
</owl:ObjectProperty>
<owl:ObjectProperty
rdf:about="http://www.example.com/merge.owl#punyakandungan">
  <rdfs:range rdf:resource="http://www.example.com/merge.owl#kandungan"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#obat"/>
</owl:ObjectProperty>
<owl:ObjectProperty
rdf:about="http://www.example.com/merge.owl#punyaobat">
  <rdfs:range rdf:resource="http://www.example.com/merge.owl#obat"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#penyakit"/>
</owl:ObjectProperty>
```

Gambar 4.34 *File ontology object property*

Dari kode program di atas *list* yang dimasukan berbentuk *nested list*, di mana untuk setiap subyek, predikat dan obyek dijadikan 1 *list*. Pada kode program di atas mengambil *listObjectProperty* kemudian melakukan perulangan sebanyak *object property* yang didapat dari *generate* data *JSON* dan *JSONSchema*-nya. Setelah membuat *object property* kemudian penulis membuat *datatype property*, Potongan kode program dan hasilnya dapat dilihat pada Gambar 4.35 dan Gambar 4.36.

```
for (int i = 0; i < listDataTypeProperty.length; i++) {
            DatatypeProperty dt = mo.createDatatypeProperty(mergeURI+
"punya" + listDataTypeProperty[i].get(1));
            dt.addDomain(mo.getResource(mergeURI +
listDataTypeProperty[i].get(0)));
}
```

Gambar 4.35 Kode program membuat *data type property*

```
<owl:DatatypeProperty rdf:about="http://www.example.com/merge.owl#punyasenyawa">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#kandungan"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.example.com/merge.owl#punyatumbuhan">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#kandungan"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.example.com/merge.owl#punyanamaO">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#obat"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.example.com/merge.owl#punyanamaP">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#penyakit"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="http://www.example.com/merge.owl#punyaidP">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#integer"/>
  <rdfs:domain rdf:resource="http://www.example.com/merge.owl#penyakit"/>
</owl:DatatypeProperty>
```

Gambar 4.36 *File ontology data type property*

Seperti sebelumnya kode program ini akan melakukan perulangan sebanyak *datatype property* yang disimpan dalam *list*. Kemudian akan memasukan ke dalam *ontology*. Selanjutnya adalah kode program untuk membuat individual dari *list* individual yang nantinya akan digunakan untuk membuat *statement*. Potongan kode program dan hasilnya dapat dilihat pada Gambar 4.37 dan Gambar 4.38.

```
    for (int i = 0; i < listIndividu.length; i++) {
            OntClass newclass = mo.getOntClass(mergeURI +
listIndividu[i].get(1));
            mo.createIndividual(mergeURI + listIndividu[i].get(0),
newclass);
        }
```

Gambar 4.37 Kode program membuat individu

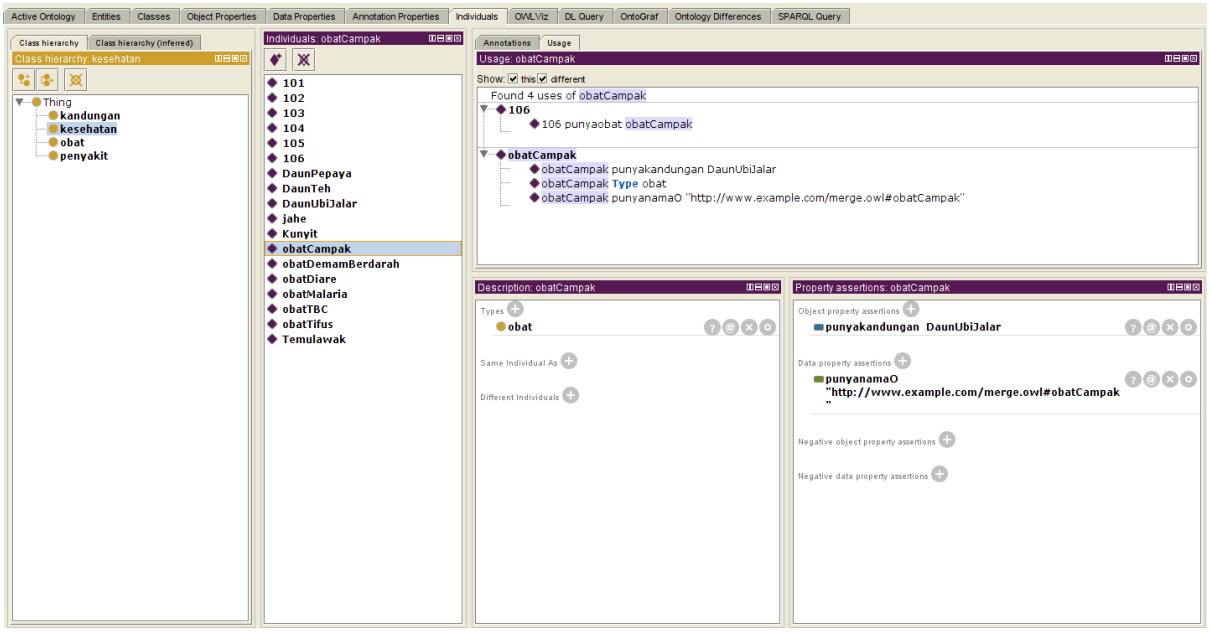
```
<base:obat rdf:about="http://www.example.com/merge.owl#obatMalaria"/>
<base:obat rdf:about="http://www.example.com/merge.owl#obatDemamBerdarah"/>
<base:kandungan rdf:about="http://www.example.com/merge.owl#DaunUbiJalar"/>
<base:penyakit rdf:about="http://www.example.com/merge.owl#102"/>
<base:penyakit rdf:about="http://www.example.com/merge.owl#104"/>
<base:penyakit rdf:about="http://www.example.com/merge.owl#106"/>
<base:obat rdf:about="http://www.example.com/merge.owl#obatCampak"/>
<base:kandungan rdf:about="http://www.example.com/merge.owl#DaunPepaya"/>
<base:kandungan rdf:about="http://www.example.com/merge.owl#Temulawak"/>
<base:obat rdf:about="http://www.example.com/merge.owl#obatTBC"/>
<base:penyakit rdf:about="http://www.example.com/merge.owl#101"/>
<base:penyakit rdf:about="http://www.example.com/merge.owl#103"/>
```

Gambar 4.38 *File ontology individual*

Setelah membuat individual dari *list* yang telah dibuat dimodul selanjutnya. Langkah selanjutnya adalah pembuatan *statement data type property* dan *statement object property.* Potongan kode program dapat dilihat pada Gambar 4.39 dan hasil dari pembuatan statement yang ditampilkan menggunakan protégé dapat dilihat pada Gambar 4.40. Protégé adalah sebuah software yang digunakan untuk mengelola ontology. Pembahasan lebih lengkap mengenai protégé dapat dilihat pada pengujian sistem.

```
for (int i = 0; i < listStatementDataTypeProperty.length; i++) {
            Literal r = mo.createLiteral(mergeURI +
listStatementDataTypeProperty[i].get(2));
            Individual ex = mo.getIndividual(mergeURI +
listStatementDataTypeProperty[i].get(0));
            Property ex1 = mo.getProperty(mergeURI +
listStatementDataTypeProperty[i].get(1));
            Statement news = mo.createStatement(ex, ex1, r);
            mo.add(news);
        }
for (int i = 0; i < listStatementObjectProperty.length; i++) {
            Resource o =
mo.createResource(mergeURI+listStatementObjectProperty[i].get(2));
            Resource ex = mo.getResource(mergeURI +
listStatementObjectProperty[i].get(0));
            Property ex1 = mo.getProperty(mergeURI +
listStatementObjectProperty[i].get(1));
            Statement news = mo.createStatement(ex, ex1, o);
            mo.add(news);}
```

Gambar 4.39 Kode program membuat statement data type property

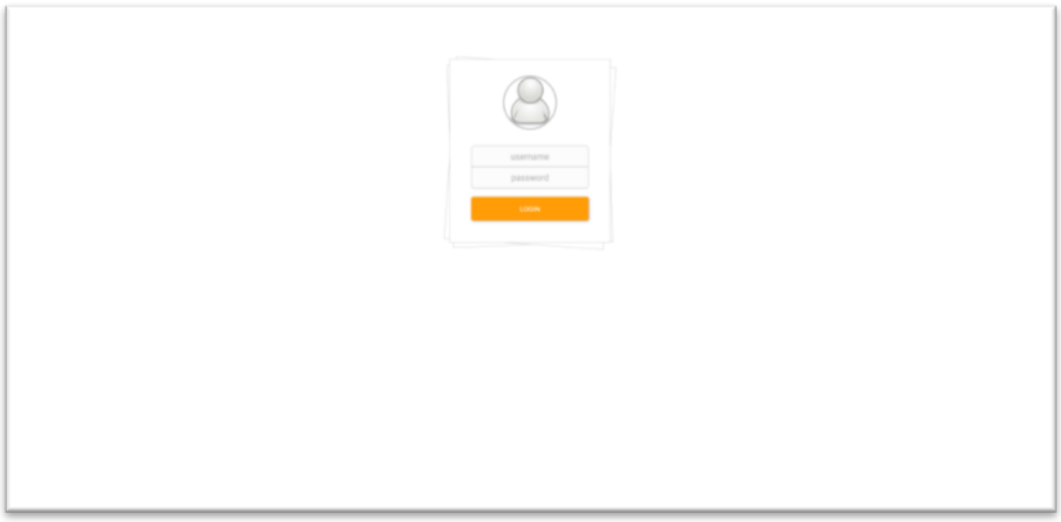


Gambar 4.40 Detail individu dan *statement*

4.2.4 Modul 4 - *interface*

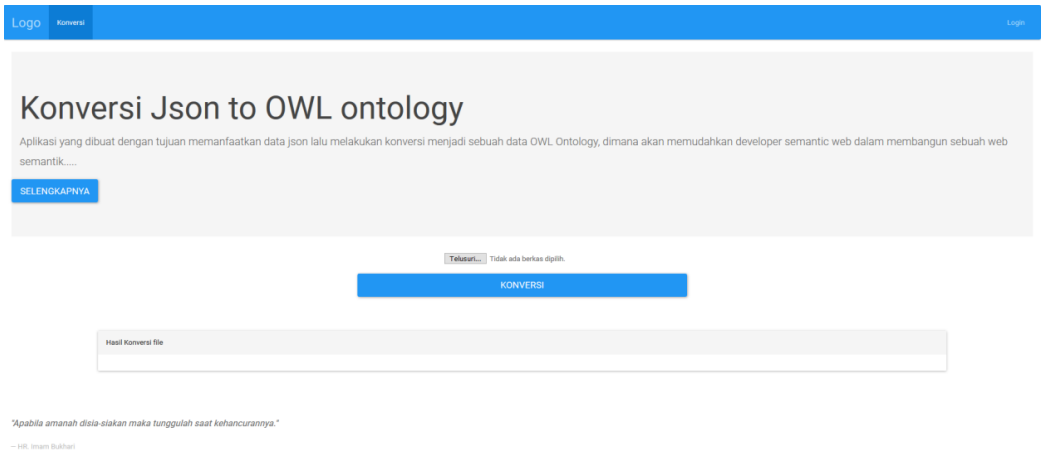
Pada modul 4 ini merupakan bagian *front end* yang akan dilihat oleh pengguna. Di mana pada modul ini adalah interface yang dapat menghandle proses yang dilakukan oleh ke 3 modul sebelumnya. Pada halaman *login* memuat informasi untuk *login* berupa *username* dan *password*. Tampilan pada halaman *login* dapat dilihat pada Gambar 4.41.

- Halaman *login*



Gambar 4.41 Halaman login

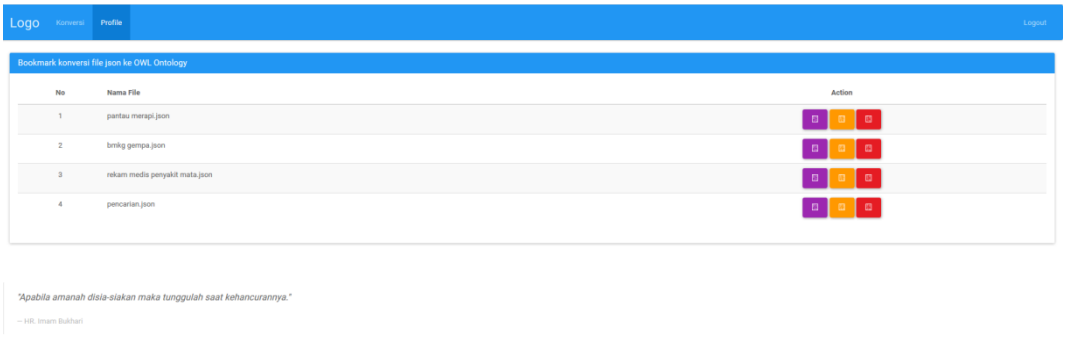
- Halaman pencarian



Gambar 4.42 Halaman awal

Halaman pencarian dapat dilihat pada Gambar 4.42. Pada halaman pencarian hanya ada 1 kotak masukan yang mana akan diisi dengan *file JSON* yang akan dikonversi. Kemudian pengguna hanya perlu menekan tombol konversi untuk melukan konversi.

- Halaman *bookmark*



Gambar 4.43 Halaman *Bookmark*

Halaman bookmark dapat dilihat pada gambar 4.43. Halaman ini menyimpan hasil *bookmark* dari hasil yang ditandai oleh pengguna. Sehingga pengguna tidak perlu melakukan konversi lagi saat dia ingin melakukan konversi data yang sama. Akan tetapi pengguna harus login terlebih dahulu sebelum menggunakan fitur ini.

4.3 Pengujian

Pada pengujian sistem kali ini. Penulis menggunakan *software* yang bernama *protégé*. Di mana software tersebut damat melakukan operasi dan membaca *file OWL Ontology*. Penulis akan menguji sistem dengan 2 *file JSON*. Kemudian penulis akan membandingkan dan menunjukan hasil yang ditampilkan (*class, Object property, Datatype property, Statement*) di *protégé*.

Dalam pengujian akan menampilkan *file JSON* masukannya. Kemudian *list* yang terbentuk dari mapping pada modul 2 dimana *list* ini akan dibandingkan dengan *property* pada *file OWL Ontology* yang dibaca dengan *protégé*. Berikut adalah pengujian sistem konversi data *JSON* ke dalam *OWL Ontology*.

4.3.1 Pengujian *File JSON*

Pada pengujian ini penulis menggunakan data yang berhubungan dengan kesehatan. di mana data tersebut memuat berbagai jenis penyakit, obat , kandungan serta senyawa yang terdapat didalamnya. Data *JSON* yang penulis gunakan untuk pengujian pertama ini adalah data yang penulis buat sendiri dengan tujuan mendapatkan data *JSON* dengan struktur yang kompleks. Sehingga dapat menguji sistem dengan lebih maksimal. potongan data *JSON* yang penulis gunakan dapat dilihat pada Gambar 4.44 dan Gambar 4.45.

- File *JSON*

```
{
  "kesehatan": {
    "penyakit": [
      {
        "idP": 101,
        "namaP": "Malaria",
        "obat": {
          "namaO": "obatMalaria",
          "kandungan": {
            "tumbuhan": "jahe",
            "senyawa": "Hcl"
          }
        }
      },
      {
        "idP": 102,
        "namaP": "demam Berdarah",
        "obat": {
          "namaO": "obatDemamBerdarah",
          "kandungan": {
            "tumbuhan": "Temulawak",
            "senyawa": "NaC20"
          }
        }
      },
      {
        "idP": 103,
        "namaP": "Diare",
        "obat": {
          "namaO": "obatDiare",
          "kandungan": {
            "tumbuhan": "DaunTeh",
            "senyawa": "Np2H"
          }
        }
      }
    ]
  }
}
```

Gambar 4.44 *File JSON*

Gambar 4.45 *File JSON*

Setelah penulis memasukan data *JSON* tersebut kedalam sistem. Sistem akan memproses data *JSON* dan dengan beberapa proses akan didapatkan *list* hasil *mapping*. Di mana *list* tersebut yang akan di-*generate* menjadi *file OWL Ontology*. Selain itu sistem juga akan menampikan detail proses dimana dalam detail proses akan membandingkan *list* yang telah didapat dari *mapping JSON* dengan *file OWL Ontology*. Dengan demikian akan menjadi acuan apakah data yang dikonversi kehilangan informasi atau tidak.

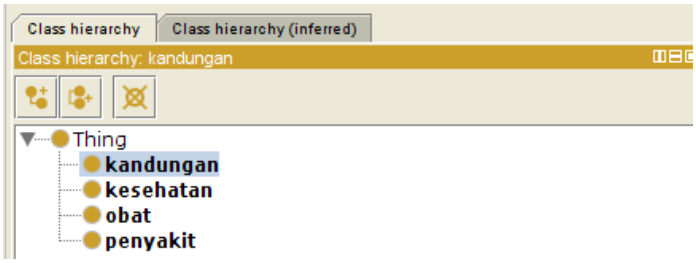
- *List Property JSON*

```
- List Class:
    [kesehatan, penyakit, obat, kandungan]

- Object Property:
    [kesehatan, punyapenyakit, penyakit]
    [penyakit, punyaobat, obat]
    [obat, punyakandungan, kandungan]
- DataType Property:
    [penyakit, idP, xsd:integer]
    [penyakit, namaP, xsd:string]
    [obat, namaO, xsd:string]
    [kandungan, tumbuhan, xsd:string]
    [kandungan, senyawa, xsd:string]

- List Individu name
    [101, penyakit]
    [102, penyakit]
    [103, penyakit]
    [obatMalaria, obat]
    [obatDemamBerdarah, obat]
    [obatDiare, obat]
    [jahe, kandungan]
    [Temulawak, kandungan]
    [DaunTeh, kandungan]

- List Statemen Datatype Property:
    [101, punyaidP, 101]
    [101, punyanamaP, Malaria]
    [102, punyaidP, 102]
    [102, punyanamaP, demam Berdarah]
    [103, punyaidP, 103]
    [103, punyanamaP, Diare]
    [obatMalaria, punyanamaO, obatMalaria]
    [obatDemamBerdarah, punyanamaO, obatDemamBerdarah]
    [obatDiare, punyanamaO, obatDiare]
    [jahe, punyatumbuhan, jahe]
    [jahe, punyasenyawa, Hcl]
    [Temulawak, punyatumbuhan, Temulawak]
    [Temulawak, punyasenyawa, NaC20]
    [DaunTeh, punyatumbuhan, DaunTeh]
    [DaunTeh, punyasenyawa, Np2H]

List Statement Object property:
    [101, punyaobat, obatMalaria]
    [102, punyaobat, obatDemamBerdarah]
    [103, punyaobat, obatDiare]
    [obatMalaria, punyakandungan, jahe]
    [obatDemamBerdarah, punyakandungan, Temulawak]
    [obatDiare, punyakandungan, DaunTeh]
```

Gambar 4.46 *List* pengujian 1

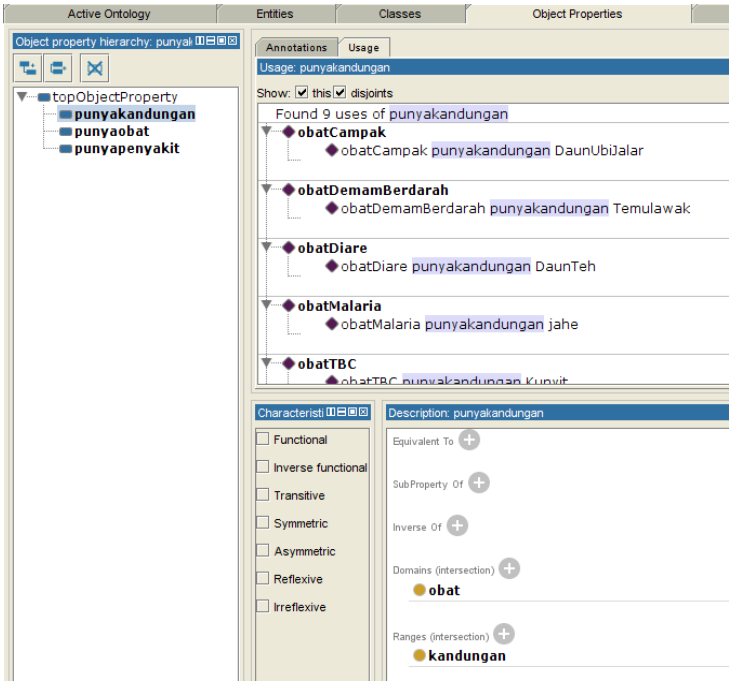
- Pengujian dengan memasukan *OWL Ontology* ke *protégé*

Protégé merupakan sebuah software yang digunakan untuk membuat sebuah domain ontology, menyesuaikan form untuk entry data, dan memasukkan data. Berbagai format penyimpanannya seperti *OWL*, *RDF*, *XML* dan *HTML*. selain itu *protégé* juga dpat digunakan untuk mengolah file *ontology*. Dengan demikian penulis akan melihat hasil dari *generate list* sebelumnya dan membandingkanya.



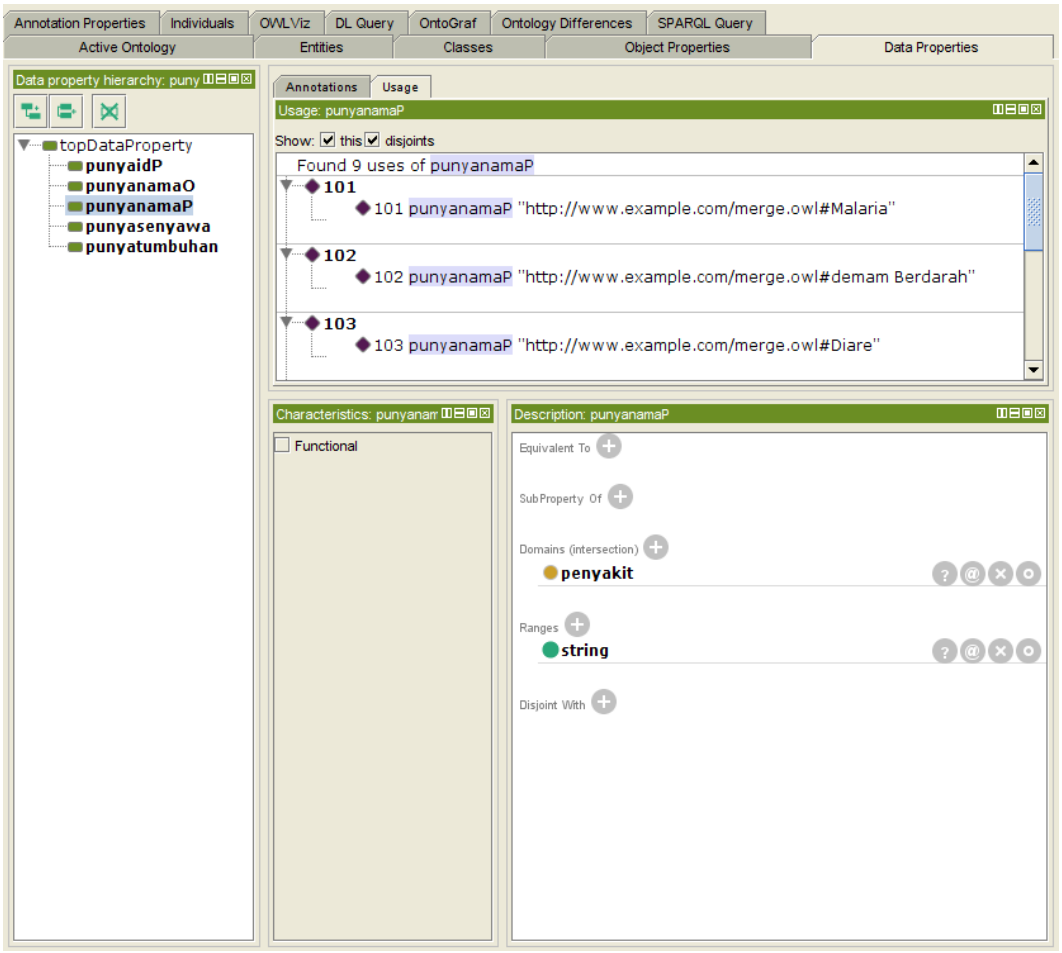
Gambar 4.47 *Protege pengujian* kelas *ontology*

Hasil *generate list class* yang didapat dari modul 3 setelah dibuka dengan menggunakan protégé dapat dilihat pada gambar 4.47. Semua isi dari *list class* sudah masuk menjadi class di *file OWL Ontology*. Dimana pada kasus ini ada 4 kelas sesuai data *JSON* yang telah dimasukan yang memiliki 4 *object.*



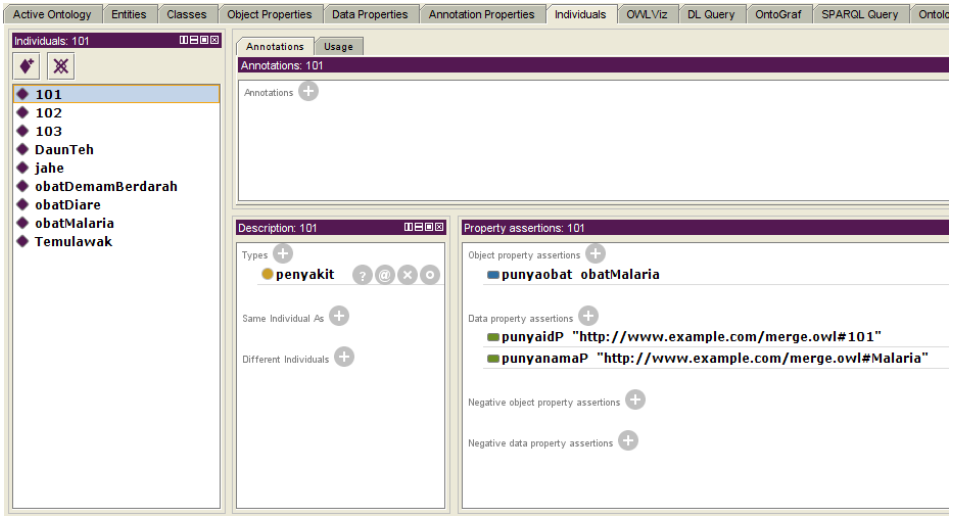
Gambar 4.48 *Protege* pengujian *object property ontology*

Hasil *generate list object property* yang didapat dari modul 3 setelah dibuka dengan menggunakan *protégé* dapat dilihat pada gambar 4.48. Semua isi dari *nested list object property* sudah masuk menjadi *object property* di *file OWL Ontology*. Dimana pada kasus ini ada 3 *list* bersarang/*nested list* sesuai data *JSON* yang telah dimasukan yang memiliki 3 *object property*. Pada setiap *list* nya memuat 3 data yaitu subjek, predikat dan objek.



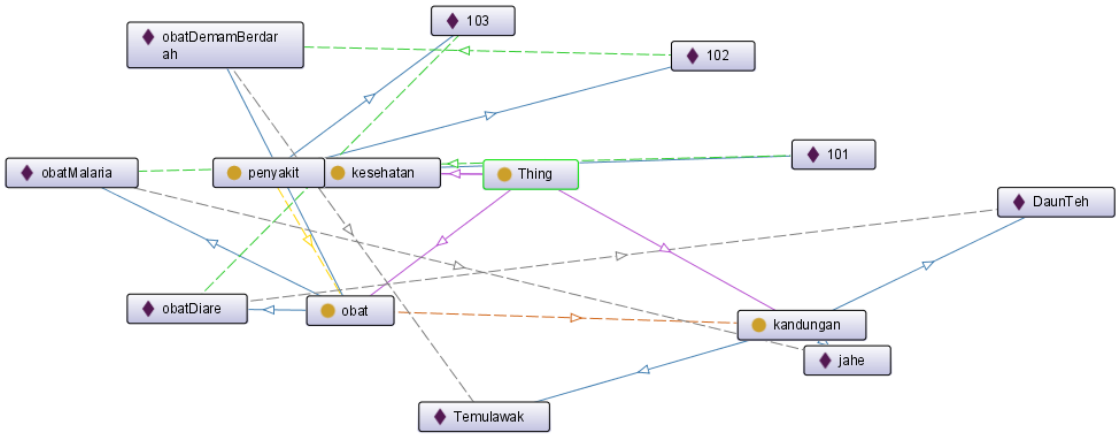
Gambar 4.49 *Protege* pengujian *data type property*

Hasil *generate list datatype property* yang didapat dari modul 3 setelah dibuka dengan menggunakan *protégé* dapat dilihat pada gambar 4.49. Semua isi dari *nested list datatype property* sudah masuk menjadi *datatype property* di *file OWL Ontology*. Dimana pada kasus ini ada 5 *list* bersarang/*nested list* sesuai data *JSON* yang telah dimasukan yang memiliki 5 *datatype property*. Pada setiap *list* nya memuat 3 data yaitu subjek, predikat dan objek.



Gambar 4.50 *Protege* pengujian individual dan statement

Hasil *generate list individual*, *statement datatype property* dan *statement object property* yang didapat dari modul 3 setelah dibuka dengan menggunakan *protégé* dapat dilihat pada gambar 4.50. Semua isi dari *nested list individual* sudah masuk menjadi individu di file *OWL Ontology*. Dimana pada kasus ini ada 9 *list* bersarang/nested *list* sesuai data *JSON* yang telah dimasukan dengan mengambil *value* pertama untuk dijadikan nama *individual* sesuai aturan *mapping* yang telah didapat dan bisa dilihat pada Tabel 3.1.



Gambar 4.51 *Protege* pengujian *ontology graph*

Dalam *protégé* terdapat fitur untuk menampilkan *ontology* kedalam bentuk *graph*. Dengan tujuan menampilkan semua *property* dalam *file ontology* untuk melihat detail dari relasi setiap *property*. *Graph file OWL Ontoloy* yang telah di-*generate* dapat dilihat pada Gambar 4.51

4.3.2 Kesimpulan pengujian

Dalam pengujian sistem ini ada beberapa parameter yang ditetapkan sebelumnya. Di mana parameter tersebut adalah *list* yang telah di*generate* harus sesuai dengan hasil yang ada pada *OWL ontology*. Sehingga konversi data *JSON* bisa dibilang "sesuai" karena tidak ada pengurangan informasi terhadap data *JSON* yang di*generate* lalu dimasukan ke dalam *OWL Ontology*.

Dalam pengujian sistem ini menggunakan data *real* dari Internet yang telah penulis cari. Akan tetapi kendala data JSON di Internet yang menggunakan bahasa Indonesia dan data bebas (legal dipakai / diakses oleh siapapun) masih tergolong sulit dan sedikit. Karena batasan masalah pada sistem ini menggunakan data *JSON* yang berbahasa Indonesia. Penulis melakukan uji coba pada 6 data *JSON* dengan Bahasa indonesia. Beberapa dari data uji yang penulis gunakan berasal dari bahasa asing, namun penulis ubah menjadi bahasa Indonesia agar sesuai dengan batasan masalah. Kemudian ada 1 data yang penulis sengaja buat sendiri untuk menguji variasi struktur *JSON*. Untuk lebih lengkap tentang data pengujian dapat dilihat pada lampiran 1 sampai lampiran 6. Tabel perbandingan *list* dengan kesesuaian hasil yang telah diuji dengan menggunakan protégé dapat dilihat pada Tabel 4.2.

Tabel 4.1 Kesimpulan pengujian

No	*List* hasil generate	Pengujian dengan protege	Perbandingan Kesesuaian
1	*List Class*	Sesuai	6/6
2	*List Object Property*	Sesuai	6/6
3	*List Data Type Property*	Sesuai	6/6
4	*List Individual*	Sesuai	6/6
5	*List Statement Data Type Property*	Sesuai	6/6
6	*List Statement Object Property*	Sesuai	6/6

Dari ke 6 data yang telah penulis uji, semua data dapat di-*mapping* oleh sistem menjadi *list* baru yang akan di-*generate* ke *OWL Ontology*. Kemudian setelah melakukan generate kedalam *OWL Ontology* dan dilakukan pengecekan menggunakan *protégé*. Semua data yang ada pada *OWL Ontology* ada di *JSON* data yang dimasukan diawal sistem dijalankan. Di mana itu menandakan bahwa data yang dikonversi dari *JSON* ke *OWL Ontology* tidak ada pengurangan informasi saat proses konversi dilakukan. Untuk detail *mapping* data *JSON* ke *OWL Ontology* nya dapat dilihat pada Tabel 3.1.

4.3.3 Kendala Yang Dialami Penulis

Dalam melakukan penelitian ini penulis mengalami kendala teknis tentang kurangnya library untuk mengolah data *JSON*. Dimana saat melakukan *research* dan proses belajar hamper setiap bahasa promrograman yang telah penulis pelajari masih sangat minimal dalam mengolah data *JSON*. Belum ada library yang lebih *advance* dalam membedah data *JSON* sehingga dalam proses *mapping* data kedalam *list* yang siap dikonversi membutuhkan *effort* lebih dalam pemrosesanya.

Kendala teknis lain adalah pada batasan masalah penelitian ini. Dimana dalam melakukan konversi sistem yang penulis kembangkan belum terdapat pemrosesan bahasa alami. Dengan modul tersebut sistem seharusnya bisa menentukan relasi secara otomatis dan lebih tepat untuk nantinya dimasukan kedalam *OWL Ontology*.

Dalam melakukan penelitian ini salah satu kendala lainya yang penulis alami adalah tentang sedikitnya referensi atau penelitian-penelitian terdahulu tentang topic yang serupa. Sehingga dalam menentukan algoritma dan proses pengerjaan penulis sedikit kesulitan pada awalnya. Karena memang belum ketemu pembahasan yang serupa.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan perancangan dan juga pengujian yang sudah dilakukan pada penelitian ini maka, didapatkan kesimpulan bahwa penelitian ini yaitu:

a. Dalam penelitian ini penulis melakukan analisis data *JSON* yang akan dikonversi kedalam *OWL Ontology*. Sehingga penulis mendapatkan tabel konversi / *mapping* data *JSON* ke *OWL Ontology*
b. Sistem yang penulis buat dapat membuat basis pengetahuan berupa *OWL Ontology* yang dapat dimanfaatkan dalam pengembangan semantic web, dengan memanfaatkan data *JSON*. Sistem akan menerima masukan berupa *file JSON* kemudian akan dilakukan proses *genetate* di *build* kembali kedalam *file OWL Ontology*

5.2 Saran

Setelah melakukan pengembangan terhadap sistem ini . penulis menyadari bahwa masih banyak kekurangan terhadap sistem yang telah dibuat ini. maka diharapkan pada penelitian selanjutnya dapat menghilangkan atau menutupi kekurangan yang ada pada sistem ini. Bahkan sangat diharapkan penelitian selanjutnya dapat mengembangkan sistem ini lebih luas lagi. Saran untuk pengembangan sistem pada penelitian selanjutnya adalah sebagai berikut:

a. Penambahan modul pemrosesan bahasa alami untuk lebih akuratnya dalam menentukan relasi setiap data hasil generate yang nantinya akan dimasukan ke dalam file *OWL* Ontology
b. Support file data selain *JSON* dengan generate Otomatis.
c. Dapat mengenali atau menganalisa *list* data yang masuk sehingga hasil yang diperoleh lebih relevan lagi

DAFTAR PUSTAKA

Benslimane, D., Dustdar, S., & Sheth, A. (2008). Services Mashups Service Mashups. Retrieved from https://91-592-722.wiki.uml.edu/file/view/services_mashups.pdf

Bray, T. (Ed.). (2014). *The JavaScript Object Notation (JSON) Data Interchange Format*. https://doi.org/10.17487/rfc7159

Fudholi, D. H. (2016). Data-Driven Dynamic Common Ontology, (August).

Gelbmann, M., & Delamer, A. (2018). Historical trends in the usage of structured data formats, October 2018. Retrieved October 22, 2018, from https://w3techs.com/technologies/history_overview/structured_data/all

Halevy, A. Y., Ives, Z. G., Mork, P., & Tatarinov, I. (2003). Piazza. *Proceedings of the Twelfth International Conference on World Wide Web - WWW '03*, *32*, 556. https://doi.org/10.1145/775152.775231

Horrocks, I., Patel-schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). SWRL : A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member Submission 21*, (May 2004), 1–20. Retrieved from https://www-old.fmi.uni-sofia.bg/Members/marian/411430437438-43e442-43743d43043d43844f-43743843c43543d-44143543c43544144244a440-2008-2009-44344743543143d430-43343e43443843d430-knowledge-bases/42344743543143d438-43c430442435440438430 43b438-course-materials

Miličić, V. (2014). Can JSON and RDF be friends? Retrieved March 7, 2018, from http://milicicvuk.com/blog/2014/08/26/can-json-and-rdf-be-friends/

Natalya F. Noy, & Deborah L. McGuinness. (2000). What is an ontology and why we need it. Retrieved March 7, 2018, from https://perso.liris.cnrs.fr/alain.mille/enseignements/Ecole_Centrale/What is an ontology and why we need it.htm

Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2015). Foundations of JSON Schema. https://doi.org/10.1145/2872427.2883029

Quan, D., Huynh, D., & Karger, D. R. (2003). LNCS 2870 - Haystack: A Platform for Authoring End User Semantic Web Applications. *LNCS*, *2870*, 738–753. Retrieved from https://link.springer.com/content/pdf/10.1007/978-3-540-39718-2_47.pdf

The Official Buzz from Blogger at Google. (2017). Blogger JSON API: Getting Started | Blogger | Google Developers. Retrieved March 7, 2018, from https://developers.google.com/blogger/docs/2.0/json/getting_started

LAMPIRAN 1

Data Uji 1

- Nama : Data Antrian RSJD Suwarjadi Jateng
- Sumber : *http://api.rsjd-sujarwadi.com/infott/*


```
{
   "umum":{
    "antrian": [
      {
        "klinik":"KLINIK_GIGI",
        "keterangan":{
          "kode_bagian": "9106",
          "sdh_panggil": 2,
          "blm_panggil": 1,
          "sdg_panggil": 2,
          "jml_pasien": 3
        }
      },
      {
        "klinik":"KLINIK_MEDIKOLEGAL",
          "keterangan":{
          "kode_bagian": "9119",
          "sdh_panggil": 18,
          "blm_panggil": 2,
          "sdg_panggil": 20,
          "jml_pasien": 20
        }
      },
      {
        "klinik":"KLINIK_TERAPI_WICARA",
        "keterangan":{
          "kode_bagian": "9415",
          "sdh_panggil": 5,
          "blm_panggil": 0,
          "sdg_panggil": 1,
          "jml_pasien": 5
        }
      },
      {
        "klinik":"KLINIK_FISIOTERAPI",
        "keterangan":{
          "kode_bagian": "9410",
          "sdh_panggil": 3,
          "blm_panggil": 0,
          "sdg_panggil": 4,
          "jml_pasien": 3
        }
      },
      {
        "klinik":"KLINIK_TKAR",
        "keterangan":{
          "kode_bagian": "9103",
```

```
          "sdh_panggil": 1,
          "blm_panggil": 0,
          "sdg_panggil": 1,
          "jml_pasien": 1
        }
      },
      {
        "klinik":"KLINIK_PSIKOGERIATRI",
        "keterangan":{
          "kode_bagian": "9118",
          "sdh_panggil": 6,
          "blm_panggil": 12,
          "sdg_panggil": 6,
          "jml_pasien": 18
        }
      },
      {
        "klinik":"KLINIK_JIWA",
        "keterangan":{
          "kode_bagian": "9101",
          "sdh_panggil": 27,
          "blm_panggil": 63,
          "sdg_panggil": 31,
          "jml_pasien": 90
        }
      },
      {
        "klinik":"KLINIK_PENYAKIT_DALAM",
        "keterangan":{
          "kode_bagian": "9109",
          "sdh_panggil": 8,
          "blm_panggil": 2,
          "sdg_panggil": 8,
          "jml_pasien": 10
        }
      }
    ]
  }
}
```

LAMPIRAN 2

Data Uji 2

- Nama : Data Kabupaten Di Provinsi Jawa Barat
- Sumber : *http://dev.farizdotid.com/api/daerahindonesia/provinsi/32/kabupaten*


```
{
  "data":{
    "daftar_kabupaten": [
      {
        "id_prov": "32",
        "informasi":{
          "id": "3201",
          "nama": "Kab.Bogor"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3202",
          "nama": "Kab.Sukabumi"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3203",
          "nama": "Kab.Cianjur"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3204",
          "nama": "Kab.Bandung"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3205",
          "nama": "Kab.HGarut"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3206",
          "nama": "Kab.Tasikmalaya"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3207",
```

```
        "nama": "Kab.Ciamis"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3208",
      "nama": "Kab.Kuningan"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3209",
        "nama": "Kab.Cirebon"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3210",
        "nama": "Kab.Majalengka"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3211",
        "nama": "Kab.Sumedang"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3212",
        "nama": "Kab.Indramayu"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3213",
        "nama": "Kab.Subang"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3214",
        "nama": "Kab.Purwakarta"
      }
    },
    {
      "id": "3215",
      "nama": "Kab.Karawang"
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3216",
```

```
        "nama": "Kab.Bekasi"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3217",
        "nama": "Kab.Bandung Barat"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3218",
        "nama": "Kab.Pangandaran"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3271",
        "nama": "Kota Bogor"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3272",
        "nama": "Kota Sukabumi"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3273",
        "nama": "Kota Bandung"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3274",
        "nama": "Kota Cirebon"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3275",
        "nama": "Kota Bekasi"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3276",
        "nama": "Kota Depok"
      }
    },
    {
```

```
        "id_prov": "32",
        "informasi":{
          "id": "3277",
          "nama": "Kota Cimahi"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3278",
          "nama": "Kota Tasikmalaya"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
            "id": "3279",
          "nama": "Kota Banjar"
        }
      }
    ]
  }
}
```

LAMPIRAN 3

Data Uji 3

- Nama : Data Kecamatan di Kabuten Bogor
- Sumber : *http://dev.farizdotid.com/api/daerahindonesia/provinsi/32/kabupaten*


```
{
  "data":{
    "daftar_kabupaten": [
      {
        "id_prov": "32",
        "informasi":{
          "id": "3201",
          "nama": "Kab.Bogor"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3202",
          "nama": "Kab.Sukabumi"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3203",
          "nama": "Kab.Cianjur"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3204",
          "nama": "Kab.Bandung"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3205",
          "nama": "Kab.HGarut"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3206",
          "nama": "Kab.Tasikmalaya"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3207",
```

```
        "nama": "Kab.Ciamis"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3208",
      "nama": "Kab.Kuningan"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3209",
        "nama": "Kab.Cirebon"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3210",
        "nama": "Kab.Majalengka"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3211",
        "nama": "Kab.Sumedang"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3212",
        "nama": "Kab.Indramayu"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3213",
        "nama": "Kab.Subang"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3214",
        "nama": "Kab.Purwakarta"
      }
    },
    {
      "id": "3215",
      "nama": "Kab.Karawang"
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3216",
```

```
        "nama": "Kab.Bekasi"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3217",
        "nama": "Kab.Bandung Barat"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3218",
        "nama": "Kab.Pangandaran"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3271",
        "nama": "Kota Bogor"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3272",
        "nama": "Kota Sukabumi"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3273",
        "nama": "Kota Bandung"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3274",
        "nama": "Kota Cirebon"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3275",
        "nama": "Kota Bekasi"
      }
    },
    {
      "id_prov": "32",
      "informasi":{
        "id": "3276",
        "nama": "Kota Depok"
      }
    },
    {
```

```
        "id_prov": "32",
        "informasi":{
          "id": "3277",
          "nama": "Kota Cimahi"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
          "id": "3278",
          "nama": "Kota Tasikmalaya"
        }
      },
      {
        "id_prov": "32",
        "informasi":{
            "id": "3279",
          "nama": "Kota Banjar"
        }
      }
    ]
  }
}
```

LAMPIRAN 4

Data Uji 4

- Nama : Data Peta Bumi Indonesia
- Sumber : *http://portal.ina-sdi.or.id*


```
{
  "map":{
    "serviceDescription":
"Peta_ini_bersumber_dari_data_RBI_Skala_1:25.000",
    "mapName": "Rupa_Bumi_Indonesia",
    "copyrightText": "Badan_Informasi_Geospasial",
    "layers": [
      {
        "id": 0,
        "informasi":{
          "name": "TRANSPORTASI",
          "parentLayerId": 1,
          "minScale": 0,
          "maxScale": 0
        }
      },
      {
        "id": 1,
        "informasi":{
          "name": "Bandara",
          "parentLayerId": 0,
          "minScale": 250000,
          "maxScale": 0
        }
      },
      {
        "id": 2,
        "informasi":{
          "name": "Light_Signal",
          "parentLayerId": 0,
          "minScale": 250000,
          "maxScale": 0
        }
      },
      {
        "id": 3,
        "informasi":{
          "name": "Tiang_dan_Pelampung_Navigasi",
          "parentLayerId": 0,
          "minScale": 250000,
          "maxScale": 0
        }
      },
      {
        "id": 4,
        "informasi":{
          "name": "Pelabuhan",
          "parentLayerId": 0,
          "minScale": 250000,
```

```
        "maxScale": 0
      }
    },
    {
      "id": 5,
      "informasi":{
        "name": "Dermaga",
        "parentLayerId": 0,
        "minScale": 250000,
        "maxScale": 0
      }
    },
    {
      "id": 6,
      "informasi":{
        "name": "Pal_Kilometer",
        "parentLayerId": 0,
        "minScale": 250000,
        "maxScale": 0
      }
    },
    {
      "id": 7,
      "informasi":{
        "name": "Ruas_Jalan",
        "parentLayerId": 0,
        "minScale": 10000000,
        "maxScale": 250001
      }
    },
    {
      "id": 8,
      "informasi":{
        "name": "Ruas_Jalan",
        "parentLayerId": 0,
        "minScale": 250000,
        "maxScale": 0
      }
    },
    {
      "id": 9,
      "informasi":{
        "name": "Jalur_Kereta_Api",
        "parentLayerId": 0,
        "minScale": 750000,
        "maxScale": 0
      }
    },
    {
      "id": 10,
      "informasi":{
        "name": "Jembatan",
        "parentLayerId": 0,
        "minScale": 250000,
        "maxScale": 0
      }
    },
    {
      "id": 11,
      "informasi":{
```

```
          "name": "Landas_Pacu",
          "parentLayerId": 0,
          "minScale": 250000,
          "maxScale": 0
        }
      },
      {
        "id": 12,
        "informasi":{
          "name": "Wilayah_Bandara",
          "parentLayerId": 0,
          "minScale": 0,
          "maxScale": 0
        }
      }
    ]
  }
}
```

LAMPIRAN 5

Data Uji 5

- Nama : Data Toko Buku
- Sumber : *https://github.com/json-path/JsonPath*,


```
{
  "toko": {
    "buku": [
      {
        "penerbit": "Erlangga",
        "informasi": {
            "kategory": "fiction",
            "penulis": "Tolkien",
            "judul": "The_Lord_of_the_Rings",
            "isbn": "0-395-19395-1",
            "harga": 2000000
            }
        },
        {
        "penerbit": "Gagas_Media.",
        "informasi": {
            "kategory": "horror",
            "penulis": "Pandu",
            "judul": "Rumah_darah",
            "isbn": "0-395-19395-2",
            "harga": 1500000
            }
        },
        {
        "penerbit": "Gramedia_Pustaka_Utama",
        "informasi": {
            "kategory": "education",
            "penulis": "bagas",
            "judul": "Matematika_dasar",
            "isbn": "0-395-19395-9",
            "harga": 70000
            }
        }
        ,{
        "penerbit": "Elexmedia_Komputindo",
        "informasi": {
            "kategory": "education",
            "penulis": "hendry",
            "judul": "GO_languange_basic",
            "isbn": "0-395-19395-8",
            "harga": 95000
            }
        }
    ]
  }
}
```

LAMPIRAN 6

Data Uji 6

- Nama : Data Penyakit dan Obat
- Sumber : Data uji yang penulis buat sendiri


```
{
  "kesehatan": {
    "penyakit": [
      {
        "idP": 101,
        "namaP": "Malaria",
        "obat": {
          "namaO": "obatMalaria",
          "kandungan": {
            "tumbuhan": "jahe",
            "senyawa": "Hcl"
          }
        }
      },
      {
        "idP": 102,
        "namaP": "demam Berdarah",
        "obat": {
          "namaO": "obatDemamBerdarah",
          "kandungan": {
            "tumbuhan": "Temulawak",
            "senyawa": "NaC20"
          }
        }
      },
      {
        "idP": 103,
        "namaP": "Diare",
        "obat": {
          "namaO": "obatDiare",
          "kandungan": {
            "tumbuhan": "DaunTeh",
            "senyawa": "Np2H"
          }
        }
      },
      {
        "idP": 104,
        "namaP": "TBC",
        "obat": {
          "namaO": "obatTBC",
          "kandungan": {
            "tumbuhan": "Kunyit",
            "senyawa": "H3P2"
          }
        }
      },
      {
        "idP": 105,
```

```
        "namaP": "Tifus",
        "obat": {
          "namaO": "obatTifus",
          "kandungan": {
            "tumbuhan": "DaunPepaya",
            "senyawa": "NaP4"
          }
        }
      },
      {
        "idP": 106,
        "namaP": "Campak",
        "obat": {
          "namaO": "obatCampak",
          "kandungan": {
            "tumbuhan": "DaunUbiJalar",
            "senyawa": "k3P2"
          }
        }
      }
    ]
  }
}
```