APLIKASI ENSIKLOPEDIA MASJID BERSEJARAH DI INDONESIA BERBASIS WEB SEMANTIK DENGAN *AUTOCOMPLETE* DAN METODE *LEVENSHTEIN DISTANCE* SEBAGAI FITUR PENCARIAN

SKRIPSI

Oleh :

ARY OKTOPIANDI PUTRADI
NIM. 11650014

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG

2017

APLIKASI ENSIKLOPEDIA MASJID BERSEJARAH DI INDONESIA BERBASIS WEB SEMANTIK DENGAN *AUTOCOMPLETE* DAN METODE *LEVENSHTEIN DISTANCE* SEBAGAI FITUR PENCARIAN

SKRIPSI

Diajukan Kepada:

Fakultas Sains dan Teknologi
Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang
Untuk Memenuhi Salah Satu Persyaratan Dalam
Memperoleh Gelar Sarjana Komputer (S. Kom)

Oleh :

ARY OKTOPIANDI PUTRADI
NIM. 11650014

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2017

APLIKASI ENSIKLOPEDIA MASJID BERSEJARAH DI INDONESIA BERBASIS WEB SEMANTIK DENGAN AUTOCOMPLETE DAN METODE LEVENSHTEIN DISTANCE SEBAGAI FITUR PENCARIAN

Oleh :

ARY OKTOPIANDI PUTRADI
NIM. 11650014

Telah disetujui oleh:

Pembimbing I	Pembimbing II
A'la Syauqi, M.Kom	Ririen Kusumawati, S.Si, M.Kom
NIP. 197712012008011007	NIP. 197203092005012002

Tanggal: Januari 2017

Mengetahui dan Mengesahkan,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 197404242009011008

APLIKASI ENSIKLOPEDIA MASJID BERSEJARAH DI INDOENSIA BERBASIS WEB SEMANTIK DENGAN AUTOCOMPLETE DAN METODE LEVENSHTEIN DISTANCE SEBAGAI FITUR PENCARIAN

SKRIPSI

Oleh :

ARY OKTOPIANDI PUTRADI
NIM. 11650014

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)

Tanggal: Januari 2017

1. Penguji Utama : Yunifa Miftachul Arif, M.T ( )
   NIP. 19830616 201101 1 004
2. Ketua : Dr. Muhammad Faisal, M.T ( )
   NIP. 19740510 200501 1 007
3. Sekretaris : A'la Syauqi, M.Kom ( )
   NIP. 19771201 200801 1 007
4. Anggota : Ririen Kusumawati, S.Si, M.Kom ( )
   NIP. 19720309 200501 2 002

Mengetahui dan Mengesahkan,
Ketua Jurusan Teknik Informatika

Dr. Cahyo Crysdian
NIP. 19740424 200901 1 008

iii

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan dibawah ini:

Nama : Ary Oktopiandi Putradi

NIM : 11650014

Jurusan : Teknik Informatika

Fakultas : Sains dan Teknologi

Menyatakan dengan sebenarnya bahwa skripsi yang saya tulis ini benar-banar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka. Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 18 Desember 2016

Yang membuat pernyataan,

Ary Oktopiandi Putradi

NIM. 11650014

iv

HALAMAN PERSEMBAHAN

Segala puji bagi Allah Swt. Kupersembahkan Skripsi ini untuk Bapak dan Mamakku tercinta, yang tiada pernah hentinya selama ini memberiku semangat, doa, dorongan, nasehat dan kasih sayang serta pengorbanan yang tak tergantikan hingga aku selalu kuat menjalani setiap rintangan yang ada didepanku.,, Bapak,.. Mamak...terimalah bukti kecil ini sebagai kado keseriusanku untuk membalas semua pengorbananmu.. dalam hidupmu demi hidupku kalian ikhlas mengorbankan segala perasaan tanpa kenal lelah.. Maafkan anakmu Bapak,,, Mamak,, masih saja ananda menyusahkanmu.

Untukmu Bapak (MUHADI),,,Mamak (MARHAINI)...Terimakasih....

(ttd.Anakmu)

Untuk adik-adikku Ika, Nita, Irma, dan Mas Rijal tiada yang paling menyenangkan saat kumpul bersama kalian, walaupun kadang bertengkar tapi hal itu selalu menjadi warna yang tak akan bisa tergantikan, terima kasih atas doa dan bantuan kalian selama ini, hanya karya kecil ini yang dapat Kak Ayik persembahkan. Maaf belum bisa menjadi panutan seutuhnya, tapi Kak Ayik akan selalu berusaha menjadi yang terbaik untuk kalian semua...

HALAMAN MOTTO

“ Janganlah kesempitan harta atau banyaknya masalah dunia, menjadikan kita putus asa terhadap rahmat Allah Swt ”

Anonim, 2017

KATA PENGANTAR

Assalamu'alaikum Wr. Wb.

Syukur alhamdulillah penulis haturkan kehadirat Allah SWT yang telah melimpahkan Rahmat dan Hidayah-Nya, sehingga penulis dapat menyelesaikan studi di Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang sekaligus menyelesaikan tugas akhir/skripsi ini dengan baik.

Selanjutnya penulis haturkan ucapan terima kasih seiring do'a dan harapan jazakumullah ahsanal jaza' kepada semua pihak yang telah membantu terselesaikannya skripsi ini. Ucapan terima kasih ini penulis sampaikan kepada:

1. Prof. Dr. H. Mudjia Raharjo, M.Si, selaku rektor UIN Maulana Malik Ibrahim Malang, yang telah banyak memberikan pengetahuan dan pengalaman yang berharga.
2. Dr. drh. Hj. Bayyinatul Muchtaromah, M.Si selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
3. Bapak Dr. Cahyo Crysdian selaku ketua Jurusan Teknik Informatika Universitas Islam Negeri Maulana Malik Ibrahim Malang.
4. Bapak A'la Syauqi, M.Kom dan Ibu Ririen Kusumawati, S. Si, M.Kom selaku dosen pembimbing skripsi, yang telah banyak memberikan pengarahan dan pengalaman yang berharga.
5. Segenap sivitas akademika Jurusan Teknik Informatika, terutama seluruh dosen, terima kasih atas segenap ilmu dan bimbingannya.
6. Ayahanda dan Ibunda serta Adik yang tercinta, yang senantiasa memberikan doa dan dukungan kepada penulis dalam menuntut ilmu.
7. Semua pihak yang ikut membantu dalam menyelesaikan skripsi ini baik berupa materil maupun moril.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih terdapat kekurangan dan penulis berharap semoga skripsi ini bisa memberikan manfaat kepada para pembaca khususnya bagi penulis secara pribadi.

Amin Ya Rabbal Alamin.

Wassalamu'alaikum Wr. Wb.

Malang, Desember 2016

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

ABSTRAK

Putradi, Ary Oktopiandi. 2017. 11650014. **Aplikasi Ensiklopedia Masjid Bersejarah di Indonesia Berbasis Web Semantik dengan Autocomplete dan Metode Levenshtein Distance Sebagai Fitur Pencarian.** Skripsi. Jurusan Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Pembimbing : (I) A'la Syauqi, M.Kom. dan (II) Ririen Kusumawati, S.Si, M.Kom.

Kata Kunci : *Web Semantik, Linked Open Data, Pencarian, Levenshtein Distance, Autocomplete*.

Informasi yang tersedia pada sebagian besar website saat ini hanya berbentuk teks dan tidak mempunyai struktur yang menyebabkan pencarian informasi hanya berdasarkan pada pencocokan teks. Untuk mengatasi masalah tersebut, maka digunakan *Web Semantik* yang memungkinkan mesin pencari untuk memahami informasi dalam website, yaitu dengan membangun *metadata* dalam graf *RDF / OWL* dan kemudian mempublikasikannya menjadi *Linked Open Data* sehingga data dapat diolah dan digunakan oleh *resource* lain untuk pertukaran data.

DBpedia adalah salah satu contoh *linked open data* yang mengekstrak informasi dari Wikipedia. Pada penelitian ini dibuat sebuah aplikasi ensiklopedia masjid bersejarah di Indonesia berbasis *web semantik* yang dapat melakukan pencarian data pada kotak pencarian dan juga dengan memilih kategori tertentu. Dan dapat melakukan pertukaran data dengan DBpedia. Untuk mengoptimalkan pencarian data pada kotak pencarian, maka pada proses pencariannya digunakan fitur *Autocomplete* dan metode *Levenshtein Distance*. *Autocomplete* berguna menampilkan sugesti jika inputan belum lengkap. Sedangkan Levenshtein Distance digunakan jika terdapat kesalahan ejaan input maka *autocomplete* akan tetap ditampilkan.

Berdasarkan hasil penelitian disimpulkan bahwa kualitas *linked open data* yang digunakan pada aplikasi mempunyai kualitas yang sangat baik dengan nilai sempurna 5 Bintang dan kinerja dari metode Levenshtein Distance dan autocomplete pada fitur pencarian mempunyai rata-rata presisi yang baik yaitu 0,8 dan mempunyai rata-rata recall sempurna yaitu 1.

ABSTRACT

Putradi, Ary Oktopiandi. 2017. 11650014. **Encyclopedia Application of Indonesian Historical Mosque Based on Semantic Web Using Autocomplete and Levenshtein Distance Method as a Search Feature.** Undergraduate Theses. Department of Informatic, Faculty of Science and Technology, The State of Islamic University Maulana Malik Ibrahim Malang.

Advisors : (I) A'la Syauqi, M.Kom. dan (II) Ririen Kusumawati, S.Si, M.Kom.

Keywords : *Semantic Web, Linked Open Data, Searching, Levenshtein Distance, Autocomplete*.

Most of searching activity at the internet based on text matching because websites are only contain of a set text and they do not have an unique structure to make a searching activity get easier and have a great accuracy. To solve this problem, Semantic Web can help the search engine to search and understand the main value of website by building metadata of RDF/OWL graf then send it into a Linked Open Data. It also makes possible to process data or use it for data exchange.

DBPedia is one of Linked Open Data which extract information from Wikipedia. On this research, we tried to make an Encyclopedia Application of Historic Mosque in Indonesia based Semantic Web in the search box and choose some categories. We also give autocomplete feature to give some suggestion output when user is typing a keyword and Levenshtein Distance will always show output search even in a wrong spelling keyword.

Base of research, it can be conclused that the quality of Linked Open Data had a good perform and Levenshtein Distance Autocomplete got 0.8 precission value. The average of reacall is 1, perfectly.

أثر فني تجريدي

بوترادي، أرى أوكتوبياندي . **تطبيق موسوعة الإندونيسية مسجد تاريخي يستند إلى الويب الدلالي باستخدام الإكمال التلقائي والأسلوب المسافة ليفينشتاين كميزة بحث.** الأطروحات الجامعية. إدارة المعلوماتية، وكلية العلوم والتكنولوجيا، والدولة من الجامعة الإسلامية مولانا إبراهيم مالك مالانغ

الكلمات الرئيسية: الويب الدلالي، ربط البيانات المفتوحة، البحث، ليفينشتاين المسافة، الإكمال التلقائي

معظم البحث في النشاط على الإنترنت على أساس النص مطابقة لأن المواقع فقط تحتوي على نص محدد، ولم يكن لديهم بنية فريدة من نوعها لتجعل من نشاط بحث أسهل، ويكون درجة كبيرة من دقة. لحل هذه المشكلة، يمكن أن تساعد "الويب الدلالي" محرك البحث البحث، وفهم أن القيمة الرئيسية للموقع عن طريق بناء البيانات الوصفية لقوات الدفاع الرواندية/البومه غراف ثم إرساله إلى "البيانات المفتوحة المرتبطة". كما أنه يجعل من الممكن لمعالجة البيانات أو استخدامها لتبادل البيانات

يديا هو واحد من "البيانات المفتوحة المرتبطة" التي استخراج المعلومات من ويكيبيديا. في هذا البحث، حاولنا أن تجعل "تطبيق موسوعة للمسجد التاريخي" في إندونيسيا على أساس الويب الدلالي في مربع البحث واختيار بعض الفئات. كما نوفر ميزة الإكمال التلقائي لإعطاء بعض الإخراج الاقتراح عند المستخدم هو كتابة الكلمة وإرادة "المسافة ليفينشتاين" دائماً إظهار الإخراج البحث حتى في كلمة خطأ إملائي.

قاعدة البحث، يمكن أن يكون كونكلوسيد أن نوعية "البيانات المفتوحة المرتبطة" أداء جيد الإكمال التلقائي لمسافة ليفينشتاين حصلت على قيمة بريسيشن 0.8. متوسط ريكال هو 1، تماما.

BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Saat ini, banyak informasi bermanfaat yang tersedia di *website*. Informasi tersebut bisa dipahami oleh manusia, tetapi mesin tidak bisa memahaminya. Misalkan ketika melakukan pencarian dengan mesin pencari, maka informasi yang ditampilkan hanya berdasarkan pada pencocokan teks dan mesin tidak mengerti dengan apa yang ditampilkan. Mesin tidak bisa menganalisa dan mengquery informasi tersebut karena berbentuk teks atau dokumen yang tidak mempunyai struktur.

Beberapa tahun lalu muncul ide untuk membuat data menjadi terstruktur, agar data pada *website* dapat dipahami oleh mesin dan seluruh data dapat terhubung secara bersamaan. Ide tersebut yaitu dengan menggunakan web semantik. Web semantik diperkenalkan pertama pada tahun 1998 ketika Sir Tim Berners Lee mempublikasi *road map* yang merangkum konsep dan idenya (Berners-Lee, 1998). Idenya memungkinkan mesin secara semantik memahami konten dalam *World Wide Web*. Ini direalisasikan dengan menambahkan *metadata* untuk menandai konten. *Metadata* berguna untuk menjelaskan dan mendeskripsikan data yang dipublikasi pada *World Wide Web* (Contoh : *RDF / OWL*). Setelah menambahkan *metadata*, konten menjadi dapat terbaca oleh mesin. Sehingga memudahkan untuk melakukan pertukaran informasi antar *website*.

Selain menambahkan *metadata*, hal yang sangat penting untuk merealisasikan web semantik yaitu tersedianya *Linked Open Data*. *Linked open*

data adalah prinsip-prinsip untuk mempublikasi data terstruktur dalam bentuk graf *RDF/OWL*, jadi data dapat digunakan oleh berbagai sumber lain. Kualitas *Linked Open Data* diperlukan sebagai pondasi untuk membuat web semantik terwujud.

Prinsip-prinsip dari *Linked Open Data* pertama dijelaskan oleh Berners-Lee pada Tahun 2006 (Berners-Lee, 2006), sebagai berikut:

1. Menggunakan *HTTP URI* sebagai pengidentifikasi dan membuatnya dapat direferensikan oleh *resources* lain.
2. Menggunakan standar terbuka *(RDF, OWL)* yang didefinisikan oleh *World Wide Web Consortium (W3C)* untuk mendeskripsikan suatu hal tertentu.
3. Memasukkan *link* ke sumber lain.

Salah satu contoh dari *Linked Open Data* yaitu DBpedia, yaitu proyek komunitas yang bertujuan untuk mengekstrak informasi dari Wikipedia dan membuatnya menjadi *Linked Open Data* pada *website* (Soeren Auer dkk, 2007).

Pada tahun 2015 dilakukan penelitian untuk mengekstrak informasi *website* sistem informasi masjid bersejarah pada alamat http://www.simas.kemenag.go.id. Peneltian ini mengekstrak informasi pada *tag HTML* pada halaman *website* masjid bersejarah menjadi data terstruktur. *Output* yang dihasilkan yaitu *file* dengan *format owl* (Zayn, 2015).

Pada penelitian ini akan dibuat aplikasi ensiklopedia masjid bersejarah di Indonesia berbasis web semantik. Data yang digunakan yaitu data yang diekstrak dari *website* http://www.simas.kemenag.go.id (Zayn, 2015) dan data DBpedia Indonesia. Dalam aplikasi ini, informasi masjid bersejarah bisa diperoleh dengan melakukan pencarian pada kotak pencarian dan juga dapat dilakukan dengan

memilih kategori tertentu. Dan jika data yang dicari sama dengan data pada DBpedia, maka data tersebut dapat digunakan sebagai informasi tambahan.

Untuk mengoptimalkan pencarian informasi masjid bersejarah maka pada proses pencariannya digunakan fitur *Autocomplete* dan metode *Levenshtein Distance*. *Autocomplete* merupakan fitur atau layanan yang dapat menampilkan prediksi kata jika kata yang diinputkan belum lengkap (Kusuma, 2012). Metode *Levenshtein Distance* digunakan jika terdapat kesalahan ejaan input beberapa huruf maka *autocomplete* akan tetap ditampilkan sehingga pencarian data lebih *user friendly* (Taleski, 2014).

1.2 IDENTIFIKASI MASALAH

1. Seberapa baik kualitas *Linked Open Data* yang digunakan dalam aplikasi.
2. Seberapa tinggi kualitas hasil pencarian fitur *Autocomplete* menggunakan metode *Levenshtein Distance*.

1.3 BATASAN MASALAH

1. Data yang digunakan adalah data hasil ekstraksi dari sistem informasi masjid bersejarah dalam format *.owl* dan juga data DBpedia Indonesia pada situs http://www.id.dbpedia.org.
2. Disebabkan masih terbatasnya layanan *Linked Open Data* yang ada, maka untuk pengujian dilakukan dengan simulasi secara lokal dengan menggunakan Fuseki Server.

1.4 TUJUAN

1. Membangun aplikasi android untuk pencarian informasi masjid bersejarah menggunakan *Linked Open Data* yang berkualitas.

2. Meningkatkan kualitas hasil pencarian data masjid pada Aplikasi Ensiklopedia Masjid Bersejarah Di Indonesia.

1.5 MANFAAT PENELITIAN

1. Memudahkan pengguna untuk mencari informasi masjid bersejarah berbasis android baik dengan menggunakan mesin pencari maupun dengan memilih kategori tertentu.
2. Memudahkan pengguna untuk mendapatkan informasi masjid bersejarah tidak hanya berasal satu sumber saja, tetapi juga berasal dari sumber lain.

1.6 SISTEMATIKA PENULISAN

Penulisan skripsi ini tersusun dalam lima bab dengan sistematika penulisan sebagai berikut :

BAB I Pendahuluan

Pendahuluan berisi latar belakang, identifikasi masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

BAB II Kajian Pustaka

Kajian pustaka berisi beberapa penelitian terdahulu yang terkait dengan penelitian ini dan juga berisi teori-teori yang mendasari dalam menyusun skripsi ini. Adapun yang dibahas dalam bab ini adalah dasar teori yang berkaitan dengan web semantik, *autocomplete*, dan *levenshtein distance*.

BAB III Metode Penelitian

Pada bab ini dijelaskan metode yang digunakan dan langkah-langkah pembuatan sistem sebagai acuan implementasi.

BAB IV Hasil dan Pembahasan

Pada bab ini dijelaskan hasil implementasi Web semantik, *Autocomplete* dan *Levenshtein Distance* pada sistem yang telah dibuat.

BAB V Penutup

Berisi Kesimpulan dari penelitian ini dan saran untuk penelitian selanjutnya.

BAB II

KAJIAN PUSTAKA

2.1 PENELITIAN TERKAIT

Sebelumnya telah dibuat aplikasi *mobile* untuk mengakses informasi biomedis seperti penyakit, gejala, obat dan sebagainya menggunakan *linked open data*. Informasi biomedis yang ditampilkan dihasilkan dengan cara mengekstrak data dari publik *open resources* yaitu *Freebase*. *Freebase* mengekstrak informasi dari Wikipedia dan dari berbagai *resources* lain. Jadi Aplikasi ini dapat menampilkan informasi biomedis dari berbagai sumber. Fitur yang ada pada aplikasi ini yaitu dapat melakukan pencarian informasi biomedis dengan mengetikkan *input* berupa istilah penyakit. Kemudian beberapa saat akan tampil beberapa daftar hasil yang relevan dengan *input*. Setelah mengklik salah satu daftar tersebut, maka akan tampil informasi berupa detail informasi, bibliografi, gambar dan video (Enrique Puertas dkk, 2013).

Selain itu telah dihasilkan aplikasi pencarian objek wisata berbasis semantik. Metode semantik yang digunakan dalam penelitian ini adalah *ontologi*. *Ontologi* berfungsi memodelkan dan menghubungkan antara entitas objek wisata dengan beberapa entitas lainnya seperti, aktifitas, agama, geografi, kategori, bangunan, dan produk wisata. Setelah dilakukan pengujian, aplikasi yang dibuat mampu mencari objek wisata berdasarkan entitas yang dimiliki seperti, aktifitas, agama, geografi, kategori, bangunan dan produk wisata. Aplikasi juga dapat melakukan klasifikasi pada objek wisata berdasarkan informasi yang sudah dimiliki serta dapat memberikan rekomendasi objek wisata (Suryawan, 2013).

Dalam penelitian ini telah dibuat perancangan sistem deteksi plagiarisme dokumen teks menggunakan algoritma *damerau levenshtein distance*. Dalam penelitiannya tersebut dilakukan pembandingan penggunaan *string matching damerau levenshtein distance* dan *levenshtein distance* serta pengaruh *stemming* menggunakan algoritma *confix stripping*. Dari hasil pengujian, *damerau levenshtein distance* membutuhkan waktu proses yang lebih lama dibanding *levenshtein distance*, sedangkan untuk nilai *similarity* yang dihasilkan relatif sama rata-rata 81 % untuk penanganan uji kasus total. Namun untuk penanganan kasus dokumen *typographical error*, *damerau levenshtein distance* dapat mengenali dokumen plagiat lebih baik (Kusuma Dkk, 2013).

Fitur *autocomplete* dan algoritma *levenshtein distance* juga pernah digunakan pada aplikasi katalog perpustakaan. Penggunaan algoritma *levenshtein distance* dalam *fitur autocomplete* berfungsi untuk menampilkan prediksi judul buku yang diketikkan pengguna sehingga mengurangi kesalahan dalam pengetikan pada kotak pencarian (Primadani, 2014).

Afta Ramadhan Zayn (2015), Aplikasi pengekstrak informasi dari *website* sebelumnya telah dibuat untuk mengekstrak informasi dari *website* sistem informasi masjid menjadi *file RDF* yang dapat dipahami oleh mesin. Proses ekstraksi menggunakan *library JSOUP* yang mempermudah dalam pengambilan data. *JSOUP* bekerja dengan cara membaca *tag-tag HTML* yang ada pada halaman situs. Sehingga diperlukan penentuan *tag HTML* untuk mengekstrak data yang diperlukan. Setelah *file* dokumen hasil ekstraksi didapat, selanjutnya dilakukan proses *generate* yang diawali dengan menentukan tempat penyimpanan hasil *generate*. Kemudian membuat *file RDF* dengan format *.owl*. Setelah *file owl*

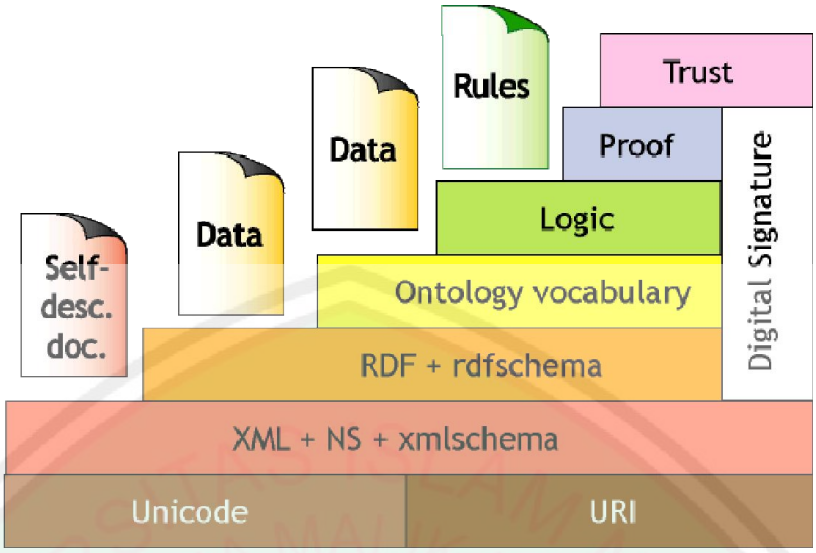
selesai dibuat, proses yang berjalan yaitu menentukan komposisi atribut *RDF*. Langkah terakhir yaitu melakukan *generate* dokumen hasil ekstraksi menjadi *file RDF* sesuai komposisi atribut *RDF* yang telah dibuat sebelumnya. Untuk mengetahui apakah *file RDF* tersebut valid maka dilakukan pengujian dengan *tool* yang terdapat pada situs https://www.w3.org/*RDF*/Validator/. Pengujian ini dilakukan dengan cara membuka *file RDF* dengan *notepad*, kemudian dilakukan pengujian isi *file rdf* dengan *tool* tersebut. Jika tidak ada kesalahan pada proses validasi maka akan ditampilkan data *web* masjid bersejarah berdasarkan komposisi atribut *RDF*.

2.2 Tinjauan Pustaka

2.2.1 Web Semantik

Informasi yang ditampilkan pada *website* biasanya didesain untuk dapat dipahami oleh manusia, ini membuat *website* sangat bergantung pada interaksi dengan manusia. Web semantik memungkinkan isi *website* dinikmati tidak hanya dalam bahasa asli pengguna, tapi juga dalam format yang bisa diakses oleh mesin atau agen-agen *software*.

Web semantik tersusun dari berbagai *layer* sebagai penyusun utama. *Layer* tersebut dapat dilihat pada **Gambar 2.1**.



Gambar 2.1. ***Layer* Web Semantik**

Layer penyusun web semantik tersebut dapat dijelaskan sebagai berikut:

1. ***URI* dan *Unicode***

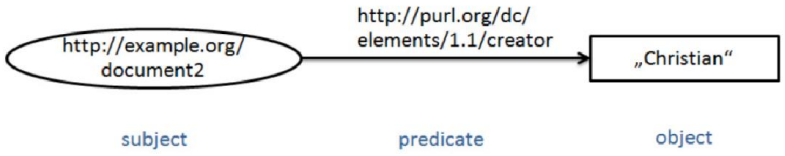
   Uniform Resource Identifier (URI) berfungsi untuk menunjukkan lokasi dan identitas dari suatu *resources*. Sedangkan *Unicode* adalah standar representasi karakter komputer.

2. ***XML* + *NS* + *XML schema***

   Berfungsi untuk menyajikan struktur data pada *website*.

3. ***RDF* dan *RDF schema***

   Resource Description Framework (RDF) adalah format data utama yang digunakan dalam web semantik. *RDF* adalah format data berbasis *triple* dimana tiap pernyataan terdiri dari subjek, predikat dan objek. Objek dari setiap *triple* bisa didefinisikan langsung (menjadikan sebuah *literal*) atau mereferensikan ke *resource* lain. *RDF-Schema (RDFS)* memberikan definisi yang digunakan oleh *RDF*. **Gambar 2.2** menunjukkan *graf RDF* sederhana yang menjelaskan pembuat dari dokumen (Weiss, 2013).



Gambar 2.2 Graf *RDF*

4. ***Ontologi Vocabulary***

Ontologi digunakan untuk mengklasifikasi informasi yang dipublikasi didalam *RDF*. Didalam *ontologi* dimungkinkan untuk membuat struktur, kamus data, dan hirarki dari *RDF*.

5. ***Logic***

Logic memungkinkan untuk membuat informasi baru berdasarkan data yang sudah ada. Ini disebut dengan *reasoning* dan terjadi dalam *logic layer* web semantik. Ada algoritma khusus dan *software tool* untuk melakukan tahapan ini.

6. ***Proof***

Proof layer digunakan untuk melacak langkah *reasoning* dan memverifikasi hasil *reasoning*.

7. ***Trust***

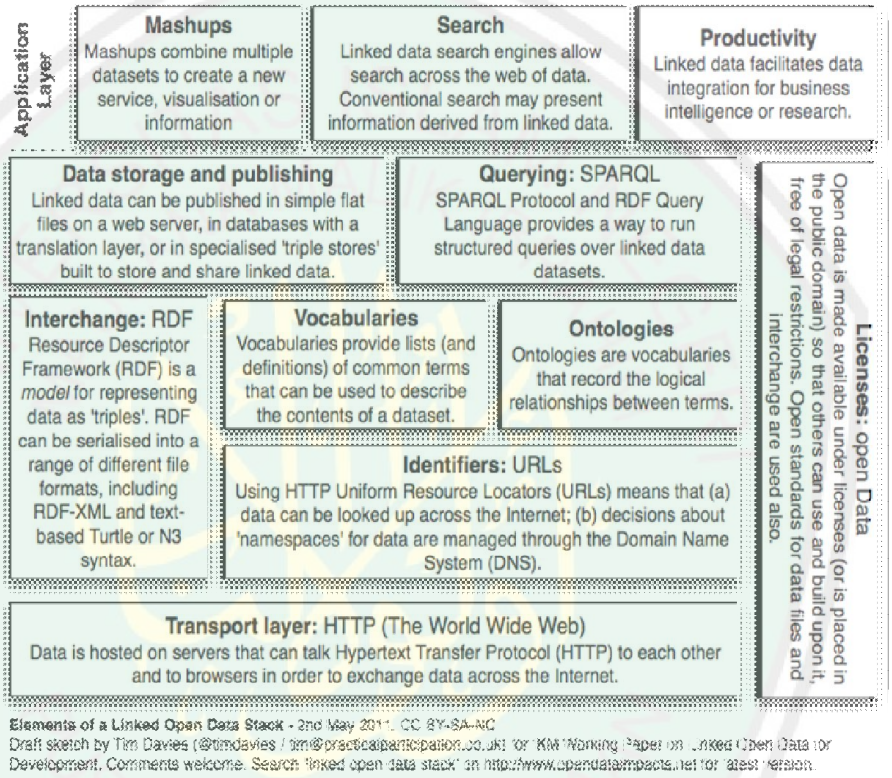
Adalah *layer* yang memungkinkan pengguna untuk mempercayai informasi yang tersaji dalam *website*.

Pada *layer* terendah (*RDF, ontologi dan logic*) sudah terstandarisasi dengan baik dan telah digunakan untuk banyak aplikasi. Pada *layer* teratas (*Proof* dan *Trust*) saat ini masih menjadi topik penelitian dan belum banyak digunakan pada aplikasi.

Linked open data (LOD) adalah prinsip-prinsip untuk mempublikasi data pada *website* secara terbuka dengan format data yang dapat dibaca oleh mesin

(*RDF / OWL*) sehingga data tersebut dapat dijadikan referensi oleh *resource* lain. Adanya *LOD* merupakan kebutuhan yang sangat penting untuk mewujudkan visi global web semantik.

Tim Davies membuat sebuah susunan teknologi *LOD* yang mirip dengan *layer* web semantik yang ditunjukkan pada **Gambar 2.3** (Davies, 2010).



Gambar 2.3 Susunan Teknologi *LOD*

Susunan *LOD* diatas adalah awal yang baik untuk memahami teknologi apa saja yang diperlukan dan bagaimana relasinya. Pada **Gambar 2.3** diatas, *HTTP* adalah protokol komunikasi yang memfasilitasi transfer informasi ke internet. *URL* digunakan sebagai pengidentifikasi suatu *resource*. *RDF* digunakan untuk petukaran data dan *ontologi* untuk menjelaskan skema antar *resource*. Data *Storage* dan *Publishing* berfungsi untuk menyimpan *file RDF* di *server* dan mempublikasinya sehingga menghasilkan sebuah *Sparql Endpoint* untuk dapat

diakses dengan berbagai *platform*. Untuk dapat mengakses *LOD* maka diperlukan *query* khusus yang disebut *Sparql Query*. *Sparql Query* adalah sebagai standar untuk mengakses *LOD*. Pada lapisan aplikasi, *LOD* dapat digunakan dalam berbagai aplikasi. Misalkan aplikasi *Mashup* yang mengkombinasikan data dari berbagai sumber untuk membuat layanan baru. Contohnya menampilkan foto sesuai dengan letak koordinat foto pada layanan *map online*. Aplikasi lain yang menggunakan *LOD* yaitu aplikasi pencarian. Aplikasi pencarian yang menggunakan *LOD* akan menghasilkan *output* pencarian yang lebih presisi karena mesin akan memproses *input* tersebut secara semantik.

Untuk menilai kualitas dari *linked open data*, Tim Berners – Lee mempublikasi sebuah artikel yang menjelaskan mengenai skema bintang untuk menilai kualitas dari *linked open data* (Berners-Lee, 2006). Skema bintang tersebut yaitu sebagai berikut:

Satu Bintang : Data tersedia pada *website* dengan lisensi terbuka dan bersifat *open data*.

Dua Bintang : Tersedia sebagai data terstruktur yang dapat dipahami mesin.

Tiga Bintang : Kedua bintang diatas ditambah dengan penggunaan format data terbuka (*CSV* bukan *Excel*).

Empat Bintang : Semua bintang diatas ditambah dengan standar terbuka dari *W3C* (*RDF* dan *SPARQL*) untuk mengidentifikasi suatu subjek.

Lima Bintang : Semua bintang diatas ditambah dengan menghubungkan data yang dimiliki dengan data dari sumber lain.

2.2.2 Fitur *Autocomplete*

Autocomplete merupakan pola yang pertama kali muncul dalam bantuan fungsi aplikasi *desktop*, dimana pengguna mengentrikan teks kedalam kotak kemudian saran pengetikan akan muncul secara otomatis (Morville & Callender, 2010).

Autocomplete memecahkan beberapa masalah umum pada pengetikan (Morville & Callender, 2010) yaitu :

a) Mengetik membutuhkan waktu
b) Pengguna tidak dapat mengeja kata dengan baik.
c) Pengguna sering bingung ketika mengetikkan kata-kata, sulit mengingat istilah yang tepat.

Autocomplete bekerja ketika pengguna menulis huruf pertama atau beberapa huruf / karakter dari sebuah kata, program yang melakukan prediksi akan mencari satu atau lebih kemungkinan kata sebagai pilihan. Jika kata yang dimaksud ada dalam pilihan kata prediksi maka kata yang dipilih tersebut akan disisipkan pada teks (Kusuma, 2012). Ilustrasi penggunaan layanan *autocomplete* dapat dilihat pada **Gambar 2.4**.

Gambar 2.4 Ilustrasi Penggunaan *Autocomplete*

2.2.3 *Levenshtein Distance*

Levenshtein distance adalah matrik untuk mengukur perbedaan antara dua *string*. *Levenshtein distance* mengukur nilai perubahan minimum sebuah karakter melalui proses penyisipan, penghapusan dan substitusi untuk mengubah satu kata menjadi kata yang lain. Dinamai sesuai penemunya Vladimir Levenshtein pada tahun 1965 (Levenshtein, 1966). Yang dimaksud dengan *distance* adalah jumlah modifikasi yang dibutuhkan untuk mengubah suatu bentuk *string* ke bentuk *string* yang lain, sebagai contoh hasil penggunaan algoritma ini, *string* "komputer" dan "*computer*" memiliki *distance* 1 karena hanya perlu dilakukan satu operasi saja untuk mengubah satu *string* ke *string* yang lain. Dalam kasus dua *string* di atas, *string* "*computer*" dapat menjadi "komputer" hanya dengan melakukan satu substitusi karakter "c" menjadi "k" (Andhika, 2010).

Algoritma *levenshtein distance* berjalan mulai dari pojok kiri atas sebuah *array* dua dimensi yang telah diisi sejumlah karakter *string* awal dan *string* target dan diberikan nilai *cost*. Jika karakter dan jumlah karakter antara *string* awal dan target sama, maka tidak dilakukan operasi perubahan. Tetapi jika karakter berbeda dengan jumlah karakter sama, maka dilakukan salah satu operasi perubahan. Nilai *cost* pada ujung kanan bawah menjadi nilai *levenshtein distance* yang menggambarkan jumlah perbedaan dua *string*.

2.2.4 Pengukuran Nilai *Similarity*

Nilai *Similarity* adalah nilai yang digunakan untuk mengetahui kemiripan antara *string input* dan *string* data. Nilai *Similarity* didefinisikan dengan persamaan 2.1 (Winoto, 2012).

$$\text{Similarity} = 1 - \frac{Levenshtein\ Distance}{Jumlah\ Huruf} \quad (2.1)$$

Dari persamaan diatas, nilai *levenshtein distance* yang telah didapatkan sebelumnya digunakan untuk menghitung nilai *similarity*.

BAB III

METODE PENELITIAN

3.1 Objek Penelitian

Pada penelitian ini akan dikembangkan sebuah aplikasi android berbasis web semantik untuk mencari informasi masjid bersejarah di Indonesia menggunakan fitur *Autocomplete* dan metode *Levenshtein Distance* sebagai fitur pencariannya.

3.2 Alat Penelitian

Dalam penelitian ini, alat penelitian terdiri dari *hardware* dan *software*.

3.2.1 *Hardware*

Hardware yang digunakan dalam penelitian ini yaitu dengan spesifikasi sebagai berikut:

1. Processor Intel(R) Core(TM) i3-2310M CPU @ 2.10 GHz
2. RAM 4.0 GB
3. Hardisk 640 GB
4. VGA Nvidia Geforce GT 540M

3.2.2 *Software*

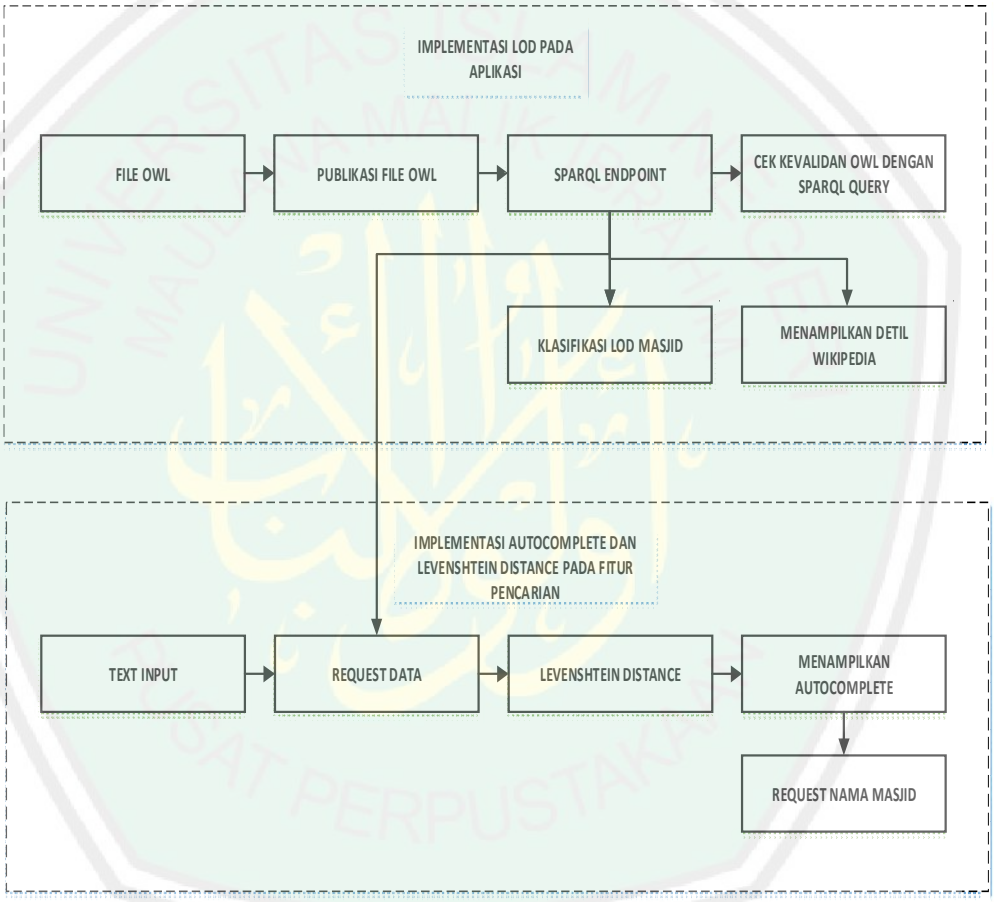
Software yang digunakan dalam penelitian ini adalah :

1. Sistem Operasi Windows 7 Ultimate
2. Apache Jena Fuseki1-1.3.0

3.3 Prosedur Penelitian

3.3.1 Desain Sistem

Desain sistem aplikasi terdiri dari dua bagian yaitu implementasi *Linked Open Data* pada aplikasi dan implementasi *Autocomplete* dan *Levenshtein Distance* pada fitur pencarian. Desain sistem tersebut ditunjukkan pada **Gambar 3.1**.



Gambar 3.1 Desain Sistem

3.3.1.1 Implementasi *Linked Open Data* (*LOD*)

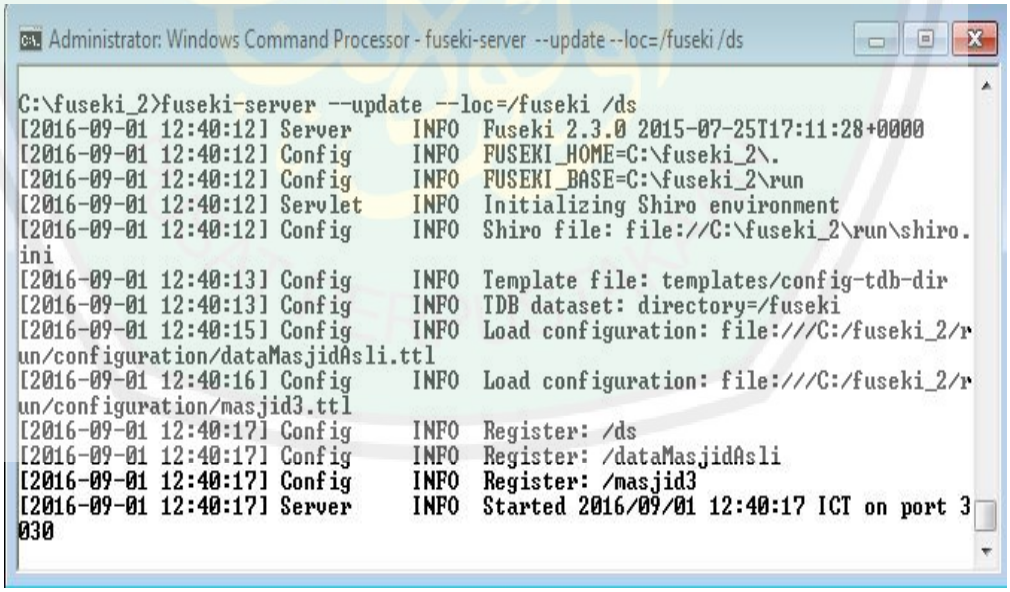
Implementasi *Linked Open Data* dimulai dengan melakukan publikasi data *owl* masjid bersejarah sehingga menjadi sebuah *SPARQL Endpoint*. Kemudian dilakukan pengecekan dengan *SPARQL Query* untuk mengetahui kevalidan *LOD*.

Jika sudah valid, maka *Linked Open Data* dapat digunakan untuk melakukan klasifikasi kategori pada aplikasi dan juga dapat dihubungkan dengan *Linked Open Data* DBpedia Indonesia.

1. **Publikasi *file owl* masjid bersejarah**

Agar data *owl* masjid bersejarah menjadi *Linked Open Data* yang dapat diakses, baik secara lokal maupun global. Maka diperlukan sebuah *SPARQL Server*. *Apache Jena Fuseki* adalah *SPARQL Server* yang memungkinkan kita untuk mempublikasi data *owl* sebagai sebuah *SPARQL endpoint* / *URL* melalui protokol *HTTP* dan diakses secara lokal.

Untuk mengaktifkan *fuseki server* digunakan *command prompt* dan perintah seperti pada **Gambar 3.2**. Gambar tersebut menandakan *fuseki server* telah aktif. *Fuseki server* dapat diakses melalui http://localhost:3030/ . Setelah itu maka akan tampil daftar dataset yang ada.



Gambar 3.2 *Command Fuseki*

Selanjutnya proses *Upload file owl* ke *fuseki server* dijelaskan dengan diagram blok pada **Gambar 3.3**. Diagram blok tersebut menjelaskan tentang proses-proses utama dalam mengupload *file owl* ke *fuseki server*. Proses tersebut dimulai dengan mengaktifkan *fuseki server* menggunakan *command* seperti pada gambar 3.1. Setelah aktif kemudian akses http://localhost:3030/ sebagai *web administrator fuseki server*. Langkah selanjutnya yaitu dengan membuat *dataset* baru, kemudian *Upload file .owl* yang tersimpan di direktori komputer. Setelah *file .owl* berhasil diupload maka akan mendapatkan sebuah *SPARQL Endpoint* sebagai *URL* untuk mengakses *LOD* yang sudah diupload ke *server*.

Gambar 3.3 Proses *Upload OWL*

2. **Pengecekan *OWL* dengan *SPARQL Query***

Pada tahap ini dilakukan pengecekan *LOD* dengan *SPARQL Query* agar dapat mengetahui apakah struktur *LOD* yang digunakan sudah valid dan dapat diakses dengan *SPARQL Query*. *SPARQL Query* adalah query untuk memanggil *Linked Open Data*. Untuk mengecek kevalidan *LOD* maka dilakukan querying pada *web administrator fuseki server* dengan cara memilih menu *query* dari

dataset yang telah dibuat. Beberapa percobaan *SPARQL Query* dapat dilihat pada **Tabel 3.1**.

Pengujian dilakukan dengan menggunakan 10 *SPARQL Query* berbeda dan semuanya berhasil menampilkan *LOD* sesuai *query* yang dibuat. Untuk menampilkan objek dengan properti tertentu maka pada awal *query* harus menggunakan *prefix* masjid, yaitu:

```
PREFIX masjid:<http://simbi.kemenag.go.id/masjid/>
```

Prefix di atas untuk menyingkat penulisan *IRI* masjid. Sehingga jika menampilkan hanya nama masjid, tidak perlu menuliskan *query* yang panjang seperti berikut:

```
SELECT ?nama WHERE
{
    ?subjek <http://simbi.kemenag.go.id/masjid/nama> ?nama
}
```

Dengan menggunakan *prefix* maka penulisan *query* menjadi lebih efektif.

Tabel 3.1 *SPARQL Query*

NO	*SPARQL QUERY*	KETERANGAN	HASIL

1	SELECT * WHERE {
	?subjek ?predikat ?objek .
	}
		Menampilkan subjek, predikat, objek dari semua *LOD* yang ada	BERHASIL

2	SELECT ?nama WHERE {
	?subjek masjid:nama ?nama .
	}
		Menampilkan *LOD* semua nama masjid	BERHASIL

3	SELECT ?kab WHERE {
	?subjek masjid:kabupaten ?kab .
	}
		Menampilkan *LOD* semua nama kabupaten	BERHASIL

4	SELECT ?nama ?kab WHERE { ?subjek masjid:nama ?nama . ?subjek	Menampilkan semua nama masjid beserta masing-masing lokasi	BERHASIL

	masjid:kabupaten ?kab } 	kabupaten tiap masjid	

5	SELECT ?nama WHERE {
	?subjek masjid:nama ?nama .?subjek masjid:kabupaten 'kab. langkat'
	}
	
		Menampilkan nama masjid yang berada di kab. langkat	BERHASIL

6	SELECT ?nama WHERE {
	?subjek masjid:nama ?nama .?subjek masjid:tahun 1933
	}
	
		Menampilkan nama masjid yang berdiri pada tahun 1933	BERHASIL

7	select ?nama where { ?subjek masjid:nama ?nama. ?subjek masjid:tahun ?tahun. FILTER (?tahun < 1900) } 	Menampilkan nama masjid yang berdiri dibawah tahun 1900	BERHASIL
8	select ?nama where { ?subjek masjid:nama ?nama. ?subjek masjid:tahun ?tahun. FILTER (?tahun >= 1900 && ?tahun <= 1950) } 	Menampilkan nama masjid yang berdiri antara tahun 1900 sampai 1950	BERHASIL

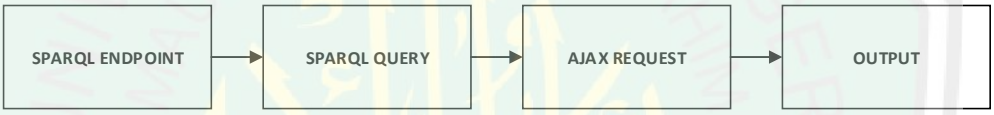
9	SELECT ?nama WHERE
	{
	{
	?subject masjid:nama ?nama. ?subject masjid:kabupaten ?kab.
	FILTER regex(?kab, '.*" + term + ".*', 'i')
	}
	}
	
		Menampilkan nama masjid berdasarkan *input* nama kabupaten pada *textfield input* yang disimpan pada variabel “term”.	BERHASIL

10	SELECT ?nama WHERE { {	Menampilkan nama masjid berdasarkan *input* rentang tahun. Mulai dari tahun pada	BERHASIL

	?subject masjid:nama ?nama. ?subject masjid:tahun ?tahun. FILTER (str (?tahun) > = '"+term+"'). FILTER (str (?tahun) < = '"+term2+"') } }	variabel "term" sampai tahun pada variabel "term2"	

3. Melakukan Klasifikasi *Linked Open Data* Masjid

Klasifikasi *LOD* masjid dilakukan dengan menggunakan *SPARQL Query* terhadap *endpoint LOD* masjid melalui *Ajax Request*. Proses Klasifikasi *LOD* masjid digambarkan dengan diagram blok pada **Gambar 3.4**.



Gambar 3.4 Diagram Blok Proses Klasifikasi

Proses Klasifikasi data masjid dimulai dengan membuat variabel endpoint untuk menampung *URL LOD* masjid, kemudian dibuat variabel untuk menampung *SPARQL Query* dan langkah terakhir yaitu *request LOD* dan *query* tersebut menggunakan *Ajax*. Untuk mengklasifikasi *LOD* masjid berdasarkan kabupaten dengan tidak ada nama kabupaten yang sama, maka digunakan *SPARQL Query* berikut:

```
SELECT distinct ?kabupaten {?subjek masjid:kabupaten
?kabupaten}
```

Dari *query* di atas, predikat masjid:kabupaten adalah *query* yang digunakan untuk mengklasifikasi *LOD* masjid.

4. Menampilkan Detil Wikipedia

Proses menampilkan detil Wikipedia berfungsi untuk menampilkan informasi tambahan dari Wikipedia jika *LOD* nama masjid yang sama tersedia juga pada Wikipedia indonesia. Proses untuk menampilkan detil Wikipedia ditunjukkan pada **Gambar 3.5**.

Gambar 3.5 Proses Menampilkan Detil Wikipedia

Untuk bisa menampilkan detil Wikipedia, maka tergantung pada *SPARQL Query* yang dibuat. *SPARQL Query* yang digunakan untuk menampilkan *LOD* dari Wikipedia yaitu dengan perintah *SERVICE*. Jadi perintah *SERVICE* digunakan untuk menggabungkan *LOD* antara dua sumber atau lebih. Contoh *Query* penggunaan perintah *SERVICE* pada **Gambar 3.6** berikut:

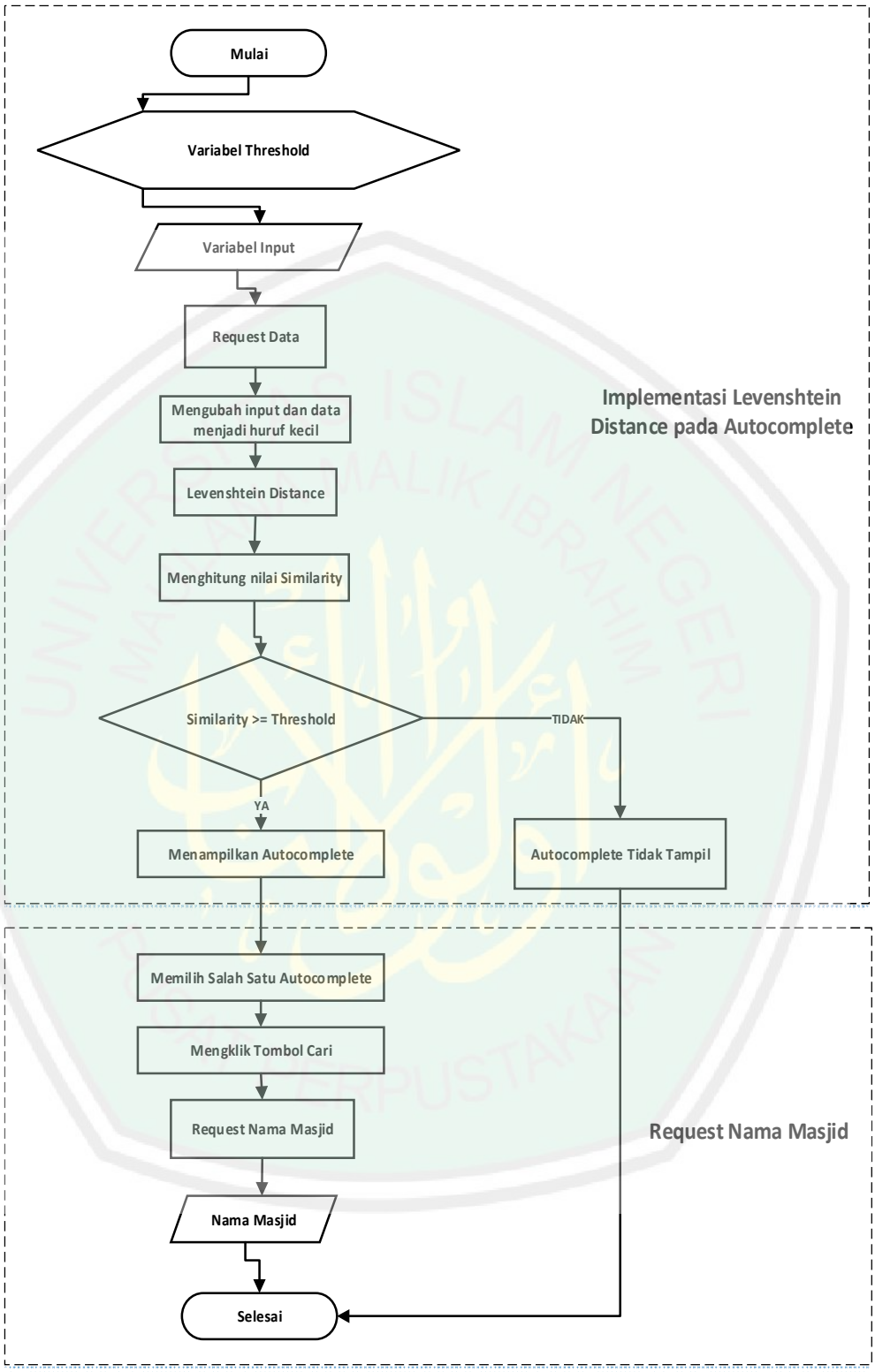
```
PREFIX masjid:<http://simbi.kemenag.go.id/masjid/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?nama ?abstrak WHERE
{
   {
   SERVICE <http://localhost:3030/masjid3>
      {
      ?s masjid:nama ?nama .
      }
   }
   BIND(STRLANG(?nama, "id") AS ?nama_id)
   SERVICE <http://id.dbpedia.org/SPARQL>
      {
      ?s2 rdfs:label ?nama_id. ?s2 rdfs:comment ?abstrak  .
      FILTER(STR(?nama_id) = ?nama)
      }
}
```

Gambar 3.6 *Query Service*

Query di atas berfungsi untuk menampilkan *LOD* nama masjid dan abstraknya berdasarkan pada sumber http://localhost:3030/masjid3 dan http://id.dbpedia.org/SPARQL.

3.3.1.2 Implementasi *Autocomplete* dan *Levenshtein Distance* pada Fitur Pencarian.

Fitur pencarian pada aplikasi ini berfungsi untuk mencari nama masjid tidak hanya berdasarkan *keyword* nama masjid, tetapi juga berdasarkan *keyword* nama kabupaten / kota dan nama kecamatan. Proses pencariannya dimulai ketika pengguna mengetikkan salah satu *keyword* nama masjid atau nama kabupaten / kota atau nama kecamatan, yang kemudian akan ditampilkan *Autocomplete* berdasarkan *keyword* yang diketikkan. Jika terjadi kesalahan pengetikkan kata, maka akan diproses dengan metode *Levenhstein Distance* sehingga *Autocomplete* akan tetap ditampilkan. Saran pencarian yang ditampilkan *Autocomplete* berdasarkan pada kemiripan antara *string input* dengan *string* data. Setelah itu pengguna memilih salah satu *Autocomplete* sebagai *keyword* dan menglik tombol cari untuk menampilkan hasil pencarian nama masjid. Hasil pencarian yang ditampilkan berdasarkan pada kata yang cocok pada *keyword*. *Flowchart* fitur pencarian nama masjid ditunjukkan pada **Gambar 3.7**. Pada gambar 3.7 tersebut, fitur pencarian nama masjid dibagi menjadi dua tahapan yaitu proses implementasi *levenshtein distance* pada *Autocomplete* dan proses *request* nama masjid.



Gambar 3.7 *Flowchart* Sistem Pencarian Nama Masjid

Adapun penjelasan dari *Flowchart* proses pencarian nama masjid di atas adalah sebagai berikut:

1. Implementasi *Levenshtein Distance* pada *Autocomplete*

a) Variabel Threshold

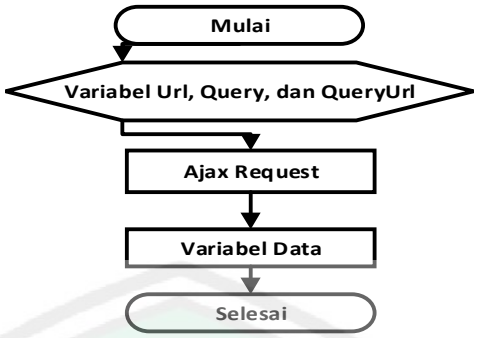
Variabel Threshold adalah variabel yang menampung nilai kemiripan minimal atau nilai ambang batas operasi perubahan / modifikasi karakter yang dilakukan. Faruq (2015) didalam penelitiannya, bahwa pencarian terbaik diperoleh dengan menggunakan level kemiripan 80%, karena mampu mengurangi kesalahan dengan baik dibanding level 90% yang hanya mampu menampilkan pencarian yang lebih sedikit, serta tidak banyak menampilkan hasil pencarian yang tidak relevan dengan kata yang diinputkan oleh *User* dibandingkaan dengan level 60% dan 70%. Jadi nilai variabel Threshold yang digunakan yaitu 80% / 100 = 0.8.

b) Variabel Input

Variabel Input adalah variabel yang menampung nilai *input* pada *textfield*. *Input* tersebut berupa nama masjid atau nama kabupaten / kota atau nama kecamatan.

c) *Request* Data

Request data berfungsi untuk mengambil data pada *server* berdasarkan *keyword* pada *textfield*. Alur dari proses *request* data ke *server* digambarkan dengan *Flowchart* pada **Gambar 3.8** berikut:



Gambar 3.8 *Flowchart Request* Data

Dari gambar 3.6 di atas, Variabel URL adalah variabel yang menampung *URL endpoint* data pada *fuseki server* yaitu http://localhost:3030/masjid. Variabel Query yaitu variabel yang menampung *SPARQL Query* terhadap *endpoint*. Variabel QueryURL adalah variabel yang menampung nilai *encode* dari variabel URL dan query, *encode* dilakukan agar *request* bisa dilakukan jika terdapat karakter-karakter khusus, misalkan karakter "#". Setelah itu dilakukan *request* ke *server* dengan data dari variabel QueryURL. *LOD* dari proses *request* disimpan dalam variabel data.

d) Mengubah Semua *String Input* dan Data Menjadi Huruf Kecil

Pengubahan *string input* dan data menjadi huruf kecil agar jika pengguna memasukkan huruf besar, maka dapat diproses dengan baik. Untuk mengubah *string input* dan data menjadi huruf kecil, maka digunakan fungsi *jquery toLowerCase()*.

e) Algoritma *Levenshtein Distance*

Pseudocode dari *Levenshtein Distance* ditunjukkan pada **Gambar 3.9**.

```
Function levenshteinDistance(string s1, string s2)
{
  int m = s1.length();


  int n = s2.length();
  for (int i = 0; i <= m; i++) {
       v[i][0] = i;
  }
  for (int j = 0; j <= n; j++) {
       v[0][j] = j;
  }

  for (int j = 1; j <= n; j++) {
       for (int i = 1; i <= m; i++) {
            if (s1[i] == s2[j])
              v[i][j] = v[i-1][j-1];
            else
              v[i][j] = minimum(v[i-1][j] + 1,
                                v[i][j-1] + 1,
                               v[i-1][j-1] + 2);
            }
       }
        return v[m][n];
}
```

Gambar 3.9 *Pseudocode Levenshtein Distance*

Algoritma *Levenshtein Distance* di atas dijelaskan pada **Tabel 3.2** berikut:

Tabel 3.2 Algoritma *Levenshtein Distance*

Langkah	Deskripsi
1	Membuat variabel m untuk menampung panjang *string* s1. Membuat variabel n untuk menampung panjang *string* s2.
2	Inisialisasi baris mulai kolom 0..m pada matrik v. Inisialisasi kolom mulai baris 0..n pada matrik v.
3	Perulangan untuk membandingkan setiap *substring* pada s2 (j dari 1 sampai n) dengan setiap *substring* pada s1 (i dari 1 sampai m).
4	Jika s1[i] = s2[j], maka nilai matrik v[i][j] = v[i-1][j-1] atau nilainya tidak berubah, yakni sama dengan nilai matrik yang berada di *diagonal* sebelah kirinya. Jika s1[i] ≠ s2[j], maka nilai matrik v[i][j] = minimum(v[i-1][j] + 1,

	v[i][j-1] + 1, v[i-1][j-1] + 2). v[i-1][j] + 1 adalah operasi hapus. v[i][j-1] + 1 adalah operasi tambah. v[i-1][j-1] + 2) adalah operasi substitusi.
5	Nilai matrik v[m,n] adalah nilai *levenshtein distance* yang dihasilkan. Nilai matrik v[m,n] adalah nilai matrik pada kolom dan baris terakhir.

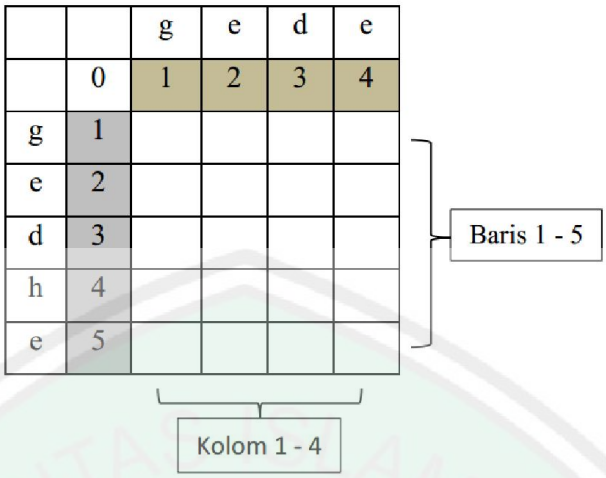
Pada bagian ini akan dijelaskan bagaimana algoritma *Levenshtein Distance* mengalami proses komputasinya dengan menggunakan matrik hingga menghasilkan nilai *Similarity*.

Contoh:

Terdapat *String Input* = gede dan *String* Data = gedhe.

Langkah-langkahnya sebagai berikut:

1. Membuat kolom matrik "gede" dengan panjang baris kurang dari panjang *string* "gedhe". Dan membuat baris matrik "gedhe" dengan panjang kolom kurang dari panjang *string* "gede". Proses pembuatan matrik ditunjukkan pada **Gambar 3.10**.



		g	e	d	e
	0	1	2	3	4
g	1				
e	2				
d	3				
h	4				
e	5				

Gambar 3.10 Proses Pembuatan Matrik

2. Melakukan pembandingan dari tiap karakter *input* dengan karakter data. Proses pembandingan karakter ditunjukkan pada **Gambar 3.11**.

		g	e	d	e
	0	1	2	3	4
g	1	0	1	2	3
e	2	1	0	1	2
d	3	2	1	0	1
h	4	3	2	1	1
e	5	4	3	2	1

Iterasi (1,1) -> Baris ke-1 Kolom ke-1

Gambar 3.11 Proses Pembandingan Karakter

a) Pada iterasi (1,1), kolom "g" sama dengan baris "g", maka nilainya tidak berubah yaitu sama dengan nilai yang berada di *diagonal* kirinya yakni 0.

b) Pada iterasi (1,2), kolom "ge" tidak sama dengan baris "g", maka huruf "e" dihapus sehingga nilainya menjadi 1.

c) Pada iterasi (1,3), kolom “ged” tidak sama dengan baris “g”, maka huruf “e”,”d” dihapus sehingga total nilainya menjadi 2.

d) Pada iterasi (1,4), kolom “gede” tidak sama dengan baris “g”, maka huruf “e”, ”d”, “e” dihapus sehingga total nilainya menjadi 3.

e) Pada iterasi (2,1), kolom “g” tidak sama dengan baris “ge”, maka huruf “e” ditambahkan sehingga nilainya menjadi 1.

f) Pada iterasi (2,2), kolom “ge” sama dengan baris “ge”, maka nilainya tidak berubah yaitu sama dengan nilai yang berada di *diagonal* kirinya yakni 0.

g) Pada iterasi (2,3), kolom “ged” tidak sama dengan baris “ge”, maka huruf “d” dihapus sehingga nilainya menjadi 1.

h) Pada iterasi (2,4), kolom “gede” tidak sama dengan baris “ge”, maka huruf “d”, “e” dihapus sehingga total nilainya menjadi 2.

i) Pada iterasi (3,1), kolom “g” tidak sama dengan baris “ged”, maka huruf “e”,”d” ditambahkan sehingga total nilainya menjadi 2.

j) Pada iterasi (3,2), kolom “ge” tidak sama dengan baris “ged”, maka huruf “d” ditambahkan sehingga nilainya menjadi 1.

k) Pada iterasi (3,3), kolom “ged” sama dengan baris “ged”, maka nilainya tidak berubah yaitu sama dengan nilai yang berada di *diagonal* kirinya yakni 0.

l) Pada iterasi (3,4), kolom “gede” tidak sama dengan baris “ged”, maka huruf ”e” dihapus sehingga nilainya menjadi 1.

m) Pada iterasi (4,1), kolom “g” tidak sama dengan baris “gedh”, maka huruf “e”, ”d”, “h” ditambahkan sehingga nilainya menjadi 3.

n) Pada iterasi (4,2), kolom “ge” tidak sama dengan baris “gedh”, maka huruf ”d”, “h” ditambahkan sehingga nilainya menjadi 2.

o) Pada iterasi (4,3), kolom “ged” tidak sama dengan baris “gedh”, maka huruf “h” ditambahkan sehingga nilainya menjadi 1.

p) Pada iterasi (4,4), kolom “gede” tidak sama dengan baris “gedh”, maka huruf “e” ditukar dengan “h” sehingga nilainya menjadi 1.

q) Pada iterasi (5,1), kolom “g” tidak sama dengan baris “gedhe”, maka huruf “e”, ”d”, “h”, “e” ditambahkan sehingga nilainya menjadi 4.

r) Pada iterasi (5,2), kolom “ge” tidak sama dengan baris “gedhe”, maka huruf ”d”, “h”, “e” ditambahkan sehingga nilainya menjadi 3.

s) Pada iterasi (5,3), kolom “ged” tidak sama dengan baris “gedhe”, maka huruf “h”, “e” ditambahkan sehingga nilainya menjadi 2.

t) Pada iterasi (5,4), kolom “gede” tidak sama dengan baris “gedhe”, maka huruf “h” ditambahkan sehingga nilainya menjadi 1.

3. Melakukan penghitungan nilai *Levenshtein Distance*. Pada gambar dibawah tergambar proses untuk mendapat nilai *Levenshtein Distance*, dimulai dengan nilai awal *Levenshtein Distance* adalah 0. Ketika jumlah karakter dan karakter yang dibandingkan sama, maka nilainya tidak berubah sesuai dengan nilai yang berada di *diagonal* kirinya. Namun jika jumlah karakter yang dibandingkan sama tetapi karakternya berbeda, maka nilainya ditambah 1 dari nilai yang berada di *diagonal* kirinya. *Levenshtein Distance* dihasilkan dari nilai matrik pada kolom dan baris terakhir, yaitu pada iterasi (5,4). Jadi nilai *Levenshtein Distance* sama

dengan 1. **Gambar 3.12** Menunjukkan proses mendapat nilai *Levenshtein Distance*.

		g	e	d	e
	0	1	2	3	4
g	1	0	1	2	3
e	2	1	0	1	2
d	3	2	1	0	1
h	4	3	2	1	1
e	5	4	3	2	1

Levenshtein Distance

Gambar 3.12 Proses mendapat nilai *Levenshtein Distance*

f) Menghitung Nilai *Similarity*

Sebelum melakukan penghitungan *Similarity*, terlebih dahulu dilakukan penghitungan jumlah huruf dari *string* "gede" dan "gedhe".

Jumlah Huruf = panjang huruf "gede" + panjang huruf "gedhe"

= 4 + 5

= 9

Setelah jumlah huruf didapatkan, kemudian dilakukan penghitungan nilai *Similarity*.

$$\text{Similarity} = 1 \quad \frac{\textit{Levenshtein Distance}}{\textit{jumlah Huruf}}$$

$$= 1 - \frac{1}{9}$$

= 8 / 9 = 0.89

Jadi Nilai *Similarity* dari *string* "gede" dan "gedhe" yaitu 0.89

g) Menampilkan *Autocomplete*

Untuk bisa menggunakan fitur *Autocomplete*, maka digunakan fungsi *jquery Autocomplete*. *Autocomplete* akan tampil jika *Similarity input* lebih dari *Threshold*.

2. *Request* Nama Masjid

a) Memilih Salah Satu *Autocomplete*

User memilih salah satu *Autocomplete* sebagai *keyword* untuk menampilkan data.

b) Mengklik Tombol Cari

User mengklik tombol cari untuk memproses *keyword* pada *textfield*.

c) *Request* Nama Masjid

Proses *request* nama masjid sama seperti proses *request* data pada gambar 3.8. Perbedaannya hanya terletak pada *query* yang digunakan. Untuk menampilkan hanya nama masjid dengan berbagai *keyword*, maka digunakan perintah *UNION*. *UNION* adalah *query* untuk mencari satu solusi diantara berbagai kemungkinan pilihan cara. Contoh Penggunaan *query UNION* pada pencarian ditunjukkan pada **Gambar 3.13**. Pada *query* tersebut dijelaskan bahwa jika menginput nama masjid atau nama kabupaten maka hasilnya adalah nama masjid.

```
SELECT ?namaMasjid
      WHERE {
                  {
                  ?subject masjid:nama  ?namaMasjid.
                  ?subject masjid:kabupaten ?kab.
                  FILTER regex(?kab, '.*" + term + ".*', 'i')
                  }
                  UNION
                  {
                  ?subject masjid:nama  ?namaMasjid.
                  FILTER regex(?namaMasjid, '.*" + term + ".*',
```

```
                    'i')
                    }
            }
```

Gambar 3.13 *Query UNION*

3.3.2 Identifikasi *Input*

Identifikasi *Input* yang ada pada sistem ditunjukkan pada **Tabel 3.3**.

Tabel 3.3 Identifikasi *Input*

Nama *Input*	Alat *Input*	Bentuk *Input*	Data yang Dimasukkan
Pencarian Nama Masjid	*Keypad*, *Touchscreen*	Teks Huruf	Nama Masjid / Nama Kabupaten / Nama Kecamatan

3.3.3 Identifikasi Proses

Proses yang ada pada sistem ini ditunjukkan pada **Tabel 3.4**.

Tabel 3.4 Identifikasi Proses

Nama Proses	Deskripsi Proses	*Input* Proses	*Output* Proses	Alur Proses
Menampilkan *Autocomplete*	Proses untuk menampilkan *Autocomplete* berdasarkan *input* pada *textfield*.	Nama Masjid / Nama Kabupaten / Nama Kecamatan	Nama Masjid / Nama Kabupaten / Nama Kecamatan	- Pilih *Tab* Beranda - *Input* nama masjid / Kabupaten / Kecamatan - Akan menampilkan *Autocomplete*
Menampilkan	Proses untuk	Nama	Nama	- Pilih Salah

Hasil Pencarian Masjid	menampilkan nama masjid berdasarkan *keyword* dari hasil *Autocomplete*.	Masjid / Nama Kabupaten / Nama Kecamatan	Masjid	satu hasil *Autocomplete* - *Tap* tombol cari
Menampilkan Kategori Kabupaten / Kota	Proses untuk mengkategorikan *LOD* masjid berdasarkan nama Kabupaten / Kota.	*Tap* tombol Kabupaten / Kota	Daftar nama kabupaten / kota	- Pilih *Tab* Kategori - *Tap* tombol Kabupaten - Menampilkan daftar nama kabupten
Menampilkan Kategori Kecamatan	Proses untuk mengkategorikan *LOD* masjid berdasarkan nama kecamatan.	*Tap* tombol Kecamatan	Daftar nama kecamatan	- Pilih *Tab* Kategori - *Tap* tombol Kecamatan - Menampilkan daftar nama kecamatan
Menampilkan Detil Masjid	Proses untuk menampilkan informasi detil masjid.	*Tap* nama masjid	Informasi Detil masjid	- *Tap* nama masjid - Menampilkan informasi detil masjid
Menampilkan detil wikipedia	Proses untuk menampilkan informasi detil masjid yang bersumber dari Wikipedia.	*Tap* tombol detil wikipedia	Informasi Detil masjid dari wikipedia	- *Tap* nama masjid - *Tap* tombol detil Wikipedia - Menampilkan

				informasi Detil masjid dari wikipedia

3.3.4 Identifikasi *Output*

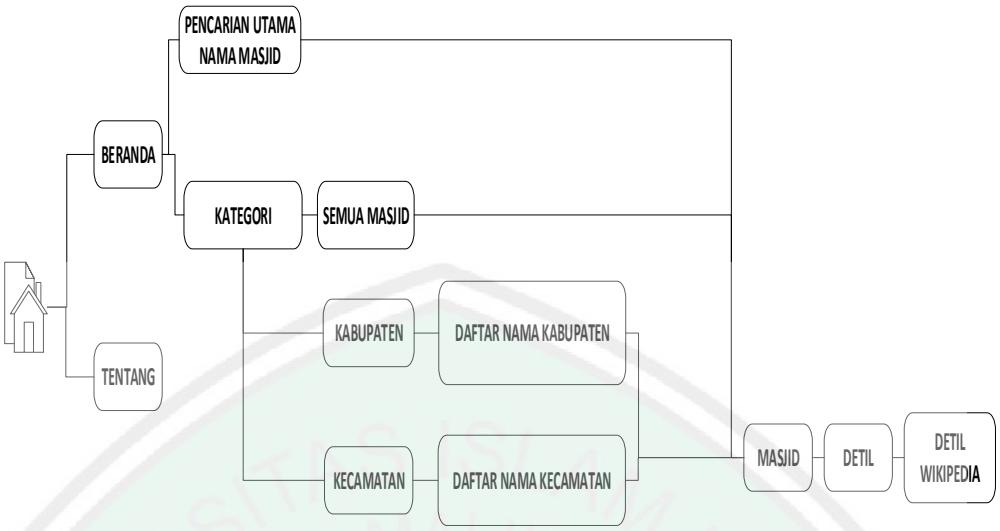
Output yang ada pada sistem ini ditunjukkan pada **Tabel 3.5**.

Tabel 3.5 Identifikasi *Output*

Nama *Output*	Alat *Output*	Bentuk *Output*	Informasi yang ditampilkan
Hasil pencarian nama masjid	*Touchscreen*	Teks	Nama Masjid
Kategori nama Kabupaten	*Touchscreen*	Teks	Nama kabupaten
Kategori nama Kecamatan	*Touchscreen*	Teks	Nama Kecamatan
Detil Masjid	*Touchscreen*	Teks	Informasi Detil masjid
Detil Wikipedia	*Touchscreen*	Teks	Informasi Detil masjid dari wikipedia

3.3.5 Desain Arsitektur

Desain arsitektur utama yang ada pada aplikasi ini terdiri dari halaman Selamat Datang yang terdiri dari Halaman Beranda dan Halaman Tentang. Halaman Beranda berisi halaman pencarian masjid beserta halaman kategori. Sedangkan halaman Tentang berisi informasi mengenai aplikasi yang dibuat. Desain arsitektur aplikasi ditunjukkan pada **Gambar 3.14.**



Gambar 3.14 Arsitektur Aplikasi

3.3.6 Desain Tampilan

Sebelum membangun tampilan yang sebenarnya dilakukan pembuatan desain tampilan terlebih dahulu. Aplikasi masjid bersejarah terdiri dari tampilan awal, tampilan beranda, tampilan tentang, tampilan kategori, tampilan masjid, tampilan kabupaten, tampilan kecamatan, tampilan detil, dan tampilan detil Wikipedia. Pada tampilan beranda terdapat fitur pencarian untuk mencari informasi masjid bersejarah di Indonesia dan juga terdapat *tab* kategori untuk mencari informasi masjid bersejarah dengan berdasarkan kateori tertentu seperti, menampilkan daftar semua nama masjid, menampilkan kategori masjid berdasar kabupaten dan menampilkan kategori masjid berdasarkan kecamatan.

1. **Desain Tampilan Awal**

Pada tampilan ini terdapat *tab* beranda untuk masuk ke tampilan beranda dan *tab* tentang untuk mengetahui tentang aplikasi yang dibuat. Desain tampilan awal ditunjukkan pada **Gambar 3.15**.

Gambar 3.15 Desain Tampilan Awal

2. Desain Tampilan Beranda

Didalam tampilan beranda terdapat fitur untuk melakukan pencarian nama masjid berdasarkan *keyword* nama masjid, kabupaten dan kecamatan yang diinput pada *textfield* serta hasil pencarian akan ditampilkan pada halaman ini. Selain *tab* beranda, terdapat juga *tab* kategori untuk masuk ke tampilan kategori ketika ingin melakukan pencarian berdasarkan kategori tertentu. Desain tampilan beranda ditunjukkan pada **Gambar 3.16**.

Gambar 3.16 Desain Tampilan Beranda

3. Desain Tampilan Kategori

Didalam tampilan kategori terdapat kategori semua masjid, kabupaten, dan kecamatan. Kategori semua masjid untuk menampilkan daftar semua masjid bersejarah yang ada. Kategori kabupaten untuk menampilkan data berdasarkan kabupaten yang ada. Kategori kecamatan untuk menampilkan data berdasarkan kecamatan yang ada. Desain tampilan kategori ditunjukkan pada **Gambar 3.17**.

Gambar 3.17 Desain Tampilan Kategori Masjid

4. Desain Tampilan Masjid

Tampilan masjid merupakan tampilan yang digunakan untuk menampilkan nama masjid bersejarah. Contohnya ketika memilih kategori semua masjid dan lain sebagainya. Desain tampilan masjid ditunjukkan pada **Gambar 3.18**.

Gambar 3.18 Desain Tampilan Masjid

5. Desain Tampilan Kabupaten

Didalam tampilan kabupaten terdapat daftar nama kabupaten dengan tidak ada nama kabupaten yang sama. Jika memilih salah satu kabupaten, maka akan ditampilkan daftar nama masjid yang berada di kabupaten yang dipilih. Desain tampilan kabupaten ditunjukkan pada **Gambar 3.19**.

Gambar 3.19 Desain Tampilan Kabupaten

6. Desain Tampilan Kecamatan

Didalam tampilan kecamatan terdapat daftar nama kecamatan dengan tidak ada nama kecamatan yang sama. Jika memilih salah satu kecamatan, maka akan ditampilkan daftar nama masjid yang berada pada kecamatan yang dipilih. Desain tampilan kecamatan ditunjukkan pada **Gambar 3.20**.

Gambar 3.20 Desain Tampilan Kecamatan

7. Desain Tampilan Detil

Tampilan detil merupakan tampilan untuk menampilkan informasi detil berdasarkan nama masjid yang dipilih. Didalam tampilan detil juga terdapat tombol untuk melihat detil dari Wikipedia Indonesia. Desain tampilan detil ditunjukkan pada **Gambar 3.21**.

Gambar 3.21 Desain Tampilan Detil

8. Desain Tampilan Detil Wikipedia

Tampilan detil Wikipedia berisi tentang informasi tambahan dari Wikipedia Indonesia. Untuk dapat masuk ke tampilan detil Wikipedia, maka dilakukan dengan mengklik tombol detil Wikipedia pada halaman detil. Desain tampilan detil Wikipedia ditunjukkan pada **Gambar 3.22**.

Gambar 3.22 Desain Tampilan Detil Wikipedia

3.3.7 Cara Pengujian Aplikasi

Cara pengujian aplikasi dibagi menjadi dua bagian yaitu pengujian kualitas *Linked Open Data* dan pengujian fitur *Autocomplete*. Pengujian kualitas *Linked Open Data* bertujuan untuk melakukan penilaian terhadap kualitas *Linked Open Data*. Sedangkan pengujian fitur *Autocomplete* bertujuan untuk mengetahui seberapa tinggi kualitas hasil pencarian fitur *Autocomplete* menggunakan metode *Levenshtein Distance*.

3.3.7.1 Pengujian Kualitas *Linked Open Data* (*LOD*)

Pengujian kualitas *Linked Open Data* dinilai berdasarkan skema bintang yang telah dibahas pada Bab II (Berners-Lee, 2006). Adapun rencana pengujiannya dapat dilihat pada **Tabel 3.6** berikut:

Tabel 3.6 Rencana Pengujian Kualitas *LOD*

No	Kualitas	Kriteria	Butir Uji
1	Satu Bintang	Data tersedia pada *web* dengan lisensi terbuka / gratis dan bersifat *open data*.	Mengakses *Linked Open Data* melalui *SPARQL Endpoint* (*URL*).
2	Dua Bintang	Tersedia sebagai data terstruktur yang dapat dipahami mesin.	Melakukan *query* menggunakan *SPARQL Query* dengan berbagai properti / predikat terhadap *Linked Open Data*.
3	Tiga Bintang	Kedua kriteria di atas ditambah dengan penggunaan format data terbuka / gratis (*CSV, JSON, XML* dll).	Melakukan *query* dengan berbagai properti / predikat terhadap *Linked Open Data* dalam berbagai format data terbuka.
4	Empat Bintang	Ketiga kriteria di atas ditambah dengan standar	Melakukan *query* terhadap subjek *linked data*

		terbuka dari *W3C* (*RDF* dan *SPARQL*) untuk mengidentifikasi suatu subjek.	menggunakan berbagai properti / predikat pada *SPARQL Query*.
5	Lima Bintang	Semua kriteria di atas ditambah dengan menghubungkan data yang dimiliki dengan data dari sumber lain.	Melakukan pencarian data nama masjid pada aplikasi, jika data yang dicari sama dengan data Wikipedia maka akan menampilkan detil informasi dari Wikipedia.

3.3.7.2 Pengujian Fitur *Autocomplete*

Dalam pengujian fitur *Autocomplete*, sistem yang diuji yaitu metode *Levenshtein Distance* pada fitur *Autocomplete*. Pengujian ini untuk mengetahui seberapa tinggi kualitas hasil pencarian fitur *Autocomplete* menggunakan metode *Levenshtein Distance* jika *User* salah mengetikkan beberapa huruf. Adapun rencana pengujiannya dapat dilihat pada **Tabel 3.7** berikut:

Tabel 3.7 Rencana Pengujian Fitur *Autocomplete*

No	Sistem yang Di Uji	Butir Uji
1	*Levenshtein Distance* pada fitur *Autocomplete*	Mengetikkan kata yang salah ejaan pada *textfield* pencarian data nama masjid

BAB IV

HASIL DAN PEMBAHASAN

Pada bab ini akan dijabarkan mengenai implementasi aplikasi berdasarkan rancangan sistem yang telah dibuat. Implementasi tersebut antara lain meliputi implementasi antarmuka atau *interface*, implementasi *web semantik*, implementasi *levenshtein distance*, dan pengujian.

4.1 Implementasi Antarmuka

Implementasi antarmuka terdiri dari tampilan awal, tampilan beranda, tampilan tentang, tampilan kategori, tampilan masjid, tampilan kabupaten, tampilan kecamatan, tampilan detil, dan tampilan detil Wikipedia.

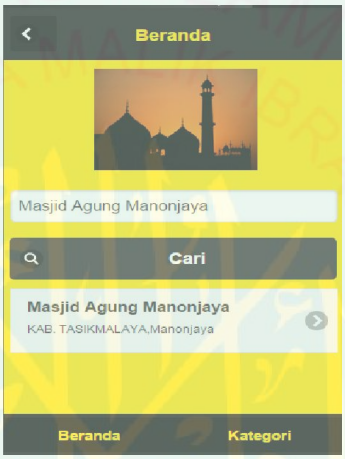
a) Tampilan Awal

Tampilan awal merupakan tampilan yang pertama kali ditampilkan ketika aplikasi dijalankan oleh *user*. Pada tampilan ini terdapat *tab* beranda untuk masuk ke tampilan beranda dan *tab* tentang untuk mengetahui tentang aplikasi yang dibuat. Tampilan awal ditunjukkan pada **Gambar 4.1**.

Gambar 4.1 Tampilan Awal

b) Tampilan Beranda

Didalam tampilan beranda terdapat fitur untuk melakukan pencarian nama masjid berdasarkan *keyword* nama masjid, kabupaten dan kecamatan yang diinput pada *textfield* serta hasil pencarian akan ditampilkan pada halaman ini. Selain *tab* beranda, terdapat juga *tab* kategori untuk masuk ke tampilan kategori ketika ingin melakukan pencarian berdasarkan kategori tertentu. Tampilan beranda ditunjukkan pada **Gambar 4.2.**



Gambar 4.2 Tampilan Beranda

c) Tampilan Tentang

Tampilan tentang menjelaskan mengenai informasi dari aplikasi yang dibuat. Tampilan tentang ditunjukkan pada **Gambar 4.3**.

d) Tampilan Kategori

Didalam tampilan kategori terdapat kategori semua masjid, kabupaten, dan kecamatan. Kategori semua masjid untuk menampilkan daftar semua masjid bersejarah yang ada. Kategori kabupaten untuk menampilkan data berdasarkan kabupaten yang ada. Kategori kecamatan untuk menampilkan data berdasarkan kecamatan yang ada. Tampilan kategori ditunjukkan pada **Gambar 4.4**.

Gambar 4.3 Tampilan Tentang

Gambar 4.4 Tampilan Kategori

e) Tampilan Masjid

Tampilan masjid merupakan tampilan yang digunakan untuk menampilkan nama masjid bersejarah. Contohnya ketika memilih kategori semua masjid dan lain sebagainya. Tampilan masjid ditunjukkan pada **Gambar 4.5**.

Gambar 4.5 Tampilan Masjid

f) Tampilan Kabupaten

Didalam tampilan kabupaten terdapat daftar nama kabupaten dengan tidak ada nama kabupaten yang sama. Jika memilih salah satu kabupaten, maka akan ditampilkan daftar nama masjid yang berada di kabupaten yang dipilih. Tampilan kabupaten ditunjukkan pada **Gambar 4.6**.

Gambar 4.6 Tampilan Kabupaten

g) Tampilan Kecamatan

Didalam tampilan kecamatan terdapat daftar nama kecamatan dengan tidak ada nama kecamatan yang sama. Jika memilih salah satu kecamatan, maka

akan ditampilkan daftar nama masjid yang berada pada kecamatan yang dipilih. Tampilan kecamatan ditunjukkan pada **Gambar 4.7**.

Gambar 4.7 Tampilan Kecamatan

h) Tampilan Detil

Tampilan detil merupakan tampilan untuk menampilkan informasi detil berdasarkan nama masjid yang dipilih. Didalam tampilan detil juga terdapat tombol untuk melihat detil dari Wikipedia Indonesia. Tampilan detil ditunjukkan pada **Gambar 4.8**.

Gambar 4.8 Tampilan Detil

i) Tampilan Detil Wikipedia

Tampilan detil Wikipedia berisi tentang informasi tambahan dari Wikipedia Indonesia. Untuk dapat masuk ke tampilan detil Wikipedia, maka dilakukan dengan mengklik tombol detil Wikipedia pada halaman detil. Tampilan detil Wikipedia ditunjukkan pada **Gambar 4.9**.

Gambar 4.9 Tampilan Detil Wikipedia

4.2 Implementasi *Linked Open Data* (*LOD*) pada Aplikasi

Pada bab sebelumnya telah dilakukan publikasi dan pengecekan kevalidan *Linked Open Data* masjid bersejarah. Setelah valid, maka *Linked Open Data* dapat digunakan untuk melakukan klasifikasi kategori pada aplikasi dan juga dapat dihubungkan dengan *Linked Open Data* DBpedia Indonesia.

4.2.1 Klasifikasi *LOD* Masjid

Proses Klasifikasi data masjid dimulai dengan membuat variabel url untuk menampung endpoint *Linked Open Data* masjid dan juga variabel query yang menampung *Sparql Query* untuk melakukan klasifikasi. **Gambar 4.10** menunjukkan potongan *Source Code* dari variabel url dan query:

```
var url = "http://127.0.0.1:3030/masjid/";


var query = "
    PREFIX masjid:<http://simbi.kemenag.go.id/masjid#>
    SELECT  distinct ?kab WHERE

    {

    ?subject  masjid:kabupaten ?kab

    }";
```

Gambar 4.10 *Source Code* Variabel Url dan Query

Query di atas adalah *query* untuk melakukan klasifikasi berdasarkan kabupaten. Untuk klasifikasi berdasarkan kecamatan, maka hanya mengubah *query* dengan predikat kecamatan. Agar *url* dan *query* yang terdiri dari banyak karakter dapat di proses dengan baik oleh *AJAX*, maka *url* dan *query* tersebut harus dilakukan *encode* terlebih dahulu. **Gambar 4.11** menunjukkan potongan *Source Code* untuk melakukan *encode*.

```
Var queryUrl = encodeURIComponent( url+"?query="+query
);
```

Gambar 4.11. Potongan *Source Code Encode Url* dan *Query*

Setelah *url* dan *query* diencode, proses selanjutnya yaitu melakukan *request* data menggunakan *AJAX*. Sintaks yang digunakan untuk melakukan *request* dengan *AJAX* yaitu $.ajax({parameter}). **Gambar 4.12** menunjukkan *Source Code AJAX request* untuk menampilkan hasil klasifikasi.

```
$.ajax
({
   url: queryUrl ,
   dataType: "json",
   success: function (result)
   {
    var i;
    var len = result.results.bindings.length;
    var list;
    var masjid;
    var masjid_kab;
    var masjid_name;
    var result_hasil = $("#result4");
    result_hasil.empty();

    for(i = 0; i < len; i++)
    {
     masjid = result.results.bindings[i];
     masjid_kab = masjid.kab.value;
     list = '<li data-kabupaten="'+masjid_kab+'"><a>';
     list += masjid_kab;
     list += '</a></li>';

     $(result_hasil).append( $(list) );
    }

    $(result_hasil).listview('refresh');
   },
    error: function (request,error)
    {
     alert(Jaringan bermasalah, silahkan coba lagi!');
    }
});
```

Gambar 4.12 *Source Code AJAX Request* untuk menampilkan Hasil Klasifikasi

Dari *Source Code AJAX request* di atas, parameter *url* digunakan untuk menentukan *URL* dalam melakukan *request* data. Parameter *datatype* digunakan untuk menentukan tipe data ketika menampilkan respon dari *server*. Parameter *success* yaitu parameter yang dijalankan ketika *request* berhasil dilakukan. Dalam parameter *success* ini, ditentukan bagaimana hasil akan ditampilkan pada aplikasi. Dan terakhir parameter *error* yaitu parameter yang dijalankan ketika *request* gagal. **Gambar 4.13** menunjukkan hasil dari klasifikasi berdasar kabupaten. Dan jika salah satu nama kabupaten dipilih maka akan menampilkan nama masjid yang

berada pada kabupaten tersebut. **Gambar 4.14** menunjukkan tampilan masjid setelah memilih salah satu kabupaten.

Gambar 4.13 Klasifikasi Berdasar Kabupaten

Gambar 4.14 Tampilan Masjid Setelah Memilih Salah Satu Kabupaten

4.2.2 Menampilkan Detil Wikipedia

Proses menampilkan detil wikipedia dimulai dengan membuat variabel url untuk menampung endpoint *Linked Open Data* masjid dan juga variabel query yang menampung *Sparql Query* untuk menampilkan data dari Wikipedia. **Gambar 4.15** menunjukkan potongan *Source Code* dari variabel url dan query.

```
var url = "http://127.0.0.1:3030/masjid/";

var query =

    "PREFIX masjid:<http://simbi.kemenag.go.id/masjid#>
     PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
     Prefix foaf: <http://xmlns.com/foaf/0.1/> ";

var query2=
    "SELECT ?nama ?kom ?img WHERE
     {
      SERVICE <http://localhost:3030/masjid>
       { ?s masjid:nama ?nama . }
     BIND(STRLANG(?nama, 'id') AS ?nama_id)
      SERVICE <http://id.dbpedia.org/sparql>
       { ?s2 rdfs:label ?nama_id. ?s2 rdfs:comment ?kom. ?s2
         foaf:depiction ?img
         FILTER(STR(?nama_id) = ?nama)
       }
     } ";
```

Gambar 4.15 Potongan *Source Code* Variabel Url dan Query

Dari gambar *Source Code* di atas, terdapat dua variabel query yang digunakan. Penggunaan dua variabel ini disebabkan oleh fungsi encodeURI yang tidak bisa melakukan *encode* karakter "#", tetapi bisa memproses perintah *SERVICE*. Maka *query* dengan terdapat karakter "#" diencode menggunakan fungsi encodeURIComponent. Variabel query2 dari Gambar 4.15 berfungsi untuk melakukan *query* terhadap dua *service* yaitu http://localhost:3030/masjid dan http://id.dbpedia.org/sparql. Karena pada *service* DBpedia terdapat tag "id" yang menunjukkan bahasa indonesia sebagai bahasa yang digunakan, maka nilai variabel ?nama dari *service* dbpedia harus diisi dalam variabel ?nama_id dengan menggunakan fungsi BIND. Proses selanjutnya yaitu melakukan *encode* dari kedua *query* tersebut. **Gambar 4.16** menunjukkan potongan *Source Code* untuk melakukan *encode* dari dua *query*.

```
var queryUrl =

url +"?query="+encodeURIComponent(query)+encodeURI(query2);
```

Gambar 4.16 Potongan *Source Code Encode* Dua *Query*

Setelah di-*encode*, kemudian dilakukan *request* data *OWL* menggunakan *AJAX* dan jika data *OWL* masjid bersejarah sama dengan data DBpedia, maka akan ditampilkan informasi dari Wikipedia pada tampilan detil wikipedia. **Gambar 4.17** menunjukkan hasil dari proses menampilkan detil Wikipedia.

Gambar 4.17 Tampilan Detil Wikipedia

4.3 Implementasi *Autocomplete* dan *Levenshtein distance* pada Fitur Pencarian

Pada aplikasi ini, fitur pencarian nama masjid dibagi menjadi dua tahapan yaitu proses implementasi *levenshtein distance* pada *Autocomplete* dan proses *request* nama masjid.

4.3.1 Implementasi *Levenshtein distance* pada *Autocomplete*

Ketika *user* menginputkan kata kunci, maka akan dilakukan *request* data *OWL* menggunakan *AJAX*. Dalam fungsi AJAX ini tidak hanya melakukan *request* data, tetapi juga memanggil fungsi autokomplit. Fungsi autokomplit yaitu fungsi yang dibuat untuk menjalankan fitur *autocomplete*. Dalam fungsi autokomplit ini, dilakukan inisialisasi *threshold*, pengubahan huruf besar menjadi kecil dan dilakukan pemanggilan fungsi levenshtein. Fungsi levenshtein adalah

fungsi yang memproses metode *levenshtein distance*. Berikut adalah langkah-langkah implementasi *levenshtein distance* pada *autocomplete*.

1. **Membangun Fungsi Levenshtein**

Metode *levenshtein distance* akan dibangun dalam sebuah fungsi dengan nama levenshtein, sehingga memudahkan untuk pemanggilannya. **Gambar 4.18** menunjukkan *Source Code* dari fungsi levenshtein.

```
function levenshtein(a, b)
{
    if(a.length === 0)  return b.length;
    if(b.length === 0)  return a.length;
    var matrix = [];

    var i;
    for(i = 0; i <= b.length; i++)
    {
      matrix[i] = [i];
    }

    var j;
    for(j = 0; j <= a.length; j++)
    {
      matrix[0][j] = j;
    }
    for(i = 1; i <= b.length; i++)
    {
      for(j = 1; j <= a.length; j++)
      {
        if(b.charAt(i-1) == a.charAt(j-1))
        {
          matrix[i][j] = matrix[i-1][j-1];
        } else
        {
          matrix[i][j] = Math.min(matrix[i-1][j-1] + 2,
          Math.min(matrix[i][j-1] + 1,
          matrix[i-1][j] + 1));
        }
      }
    }
    var dist = matrix[b.length][a.length];
    var sumLen = a.length + b.length;
    return (sumLen - dist) / sumLen;
}
```

Gambar 4.18 *Source Code* Fungsi Levenshtein

Pada fungsi levenshtein tersebut, variabel a merupakan *input user* dan variabel b merupakan kata kunci pada data *owl* masjid. Langkah pertama pada

fungsi ini yaitu terdapat kondisi untuk memastikan jika salah satu atau kedua variabel mempunyai panjang *string* bernilai 0 maka nilainya adalah variabel sebaliknya. Langkah selanjutnya yaitu melakukan perulangan baris untuk menentukan panjang matrik dan perulangan kolom untuk menentukan lebar matrik. Setelah itu dilakukan pencocokan karakter antara variabel a dan variabel b, jika karakter cocok maka nilai *cost* adalah 0, jika dilakukan operasi tambah / hapus maka nilai *cost* adalah 1, dan jika dilakukan operasi substitusi maka nilai *cost* adalah 2. Nilai *cost* pada baris dan kolom terakhir adalah nilai *levenshtein distance* yang dihasilkan. Proses selanjutnya menghitung jumlah karakter dari kedua variabel. Nilai *levenshtein distance* dan jumlah karakter kedua variabel yang diperoleh akan digunakan pada proses akhir, yaitu menghitung nilai *similarity*.

2. Membuat Fungsi Autokomplit

Fungsi autokomplit adalah fungsi yang berguna untuk menampilkan data *owl* masjid pada fitur *autocomplete* ketika melakukan pencarian data. Fungsi autokomplit ini ditunjukkan pada **Gambar 4.19**. Pada fungsi ini variabel result adalah variabel *array* data *owl* masjid yang digunakan. Variabel threshold adalah variabel yang menampung nilai *threshold*. *Method jquery autocomplete* adalah *method* untuk menjalankan fitur *autocomplete*. Dalam *method* ini terdapat *option source* dalam bentuk fungsi yang berguna untuk menghubungkan data dengan *input* pada *autocomplete*. Terdapat dua parameter yang digunakan pada fungsi *source*, pertama yaitu parameter *request* yang merujuk pada nilai *input* dan kedua yaitu parameter *response* yang merujuk pada data yang akan ditampilkan pada *autocomplete*. Pada fungsi *source* terdapat variabel userInput yang menampung

nilai *input* yang telah diubah menjadi huruf kecil, serta variabel suggestions yang menampung *array* data saran yang akan ditampilkan pada *autocomplete*. Proses pertama pada fungsi *source* yaitu dilakukan pengubahan *user input* menjadi huruf kecil. Kemudian terdapat *method forEach()* yang berfungsi untuk menjalankan sebuah fungsi terhadap setiap elemen *array* data *owl* masjid. Dalam *method* ini *array* data diubah menjadi huruf kecil, kemudian terdapat dua kondisi untuk mengisi *array* data saran. Kondisi pertama untuk menampilkan saran *autocomplete* jika *user* mulai menginput satu huruf yang sesuai dengan data *owl* masjid. Dan kondisi kedua adalah kondisi untuk menampilkan saran *autocomplete* jika *similarity levenshtein distance* lebih dari sama dengan *threshold* yang ditetapkan. Proses akhir dari fungsi *source* ini adalah menampilkan saran *autocomplete* berdasarkan data yang sesuai dengan kondisi yang ada.

```
function autokomplit()
{
    var result=ajaxResult;
    console.log(result);
    var threshold = 0.8;
    $( "#searchid" ).autocomplete(
    {
        source: function(request, response)
        {
            if ( result === undefined )
            {
                response([]);
                return;
            }
            var userInput = request.term.toLowerCase();
            var suggestions = [];

            result.forEach (function(tag)
            {
                var tagLower = tag.toLowerCase();

                if ( tagLower.indexOf(userInput)>=0 )
                {
                    suggestions.push(tag);
                } else if (levenshtein(userInput, tagLower)
                  >= threshold)
                {
                    suggestions.push(tag);
                }
            });
            response(suggestions);
        }
    });
}
```

Gambar 4.19 *Source Code* Fungsi Autokomplit

3. *Request* Data *OWL* dengan *AJAX*

Fungsi autokomplit yang telah dibuat sebelumnya dan berfungsi untuk menampilkan saran *autocomplete* tidak akan berjalan ketika tidak dilakukan *request* data dengan *AJAX*. *Request* data *owl* dengan *AJAX* merupakan proses untuk melakukan *request* data *owl* masjid sekaligus mamanggil fungsi autokomplit sehingga dapat menampilkan saran *autocomplete* yang sesuai dengan *input user*. Sebelum melakukan *request* data *owl*, terlebih dahulu dibuat variabel-variabel yang dibutuhkan untuk melakukan *request* data. Variabel-variabel tersebut adalah variabel url, variabel query, variabel queryUrl, dan variabel

ajaxResult. Variabel url berguna untuk menampung *url owl* masjid, variabel query berguna untuk menampung *query* data yang ditampilkan *autocomplete*, variabel queryUrl berguna untuk melakukan *encode* variabel query dan variabel ajaxResult adalah variabel yang menampung respon dari *request* data dalam bentuk *array*. **Gambar 4.20** menunjukkan potongan *Source Code* dari variabel url, variabel query, variabel queryUrl, dan variabel ajaxResult :

```
var url = "http://127.0.0.1:3030/masjid/";

var query =
" PREFIX masjid:<http://simbi.kemenag.go.id/masjid#>
  SELECT ?object (SAMPLE(?kab) AS ?kabupaten) (SAMPLE(?kec) as
?kecamatan)   WHERE
{
?subject masjid:nama ?object. ?subject masjid:kabupaten ?kab.
?subject masjid:kecamatan ?kec
} GROUP BY ?object ";

var queryUrl = url + "?query="+encodeURIComponent(query);
var ajaxResult=[];
```

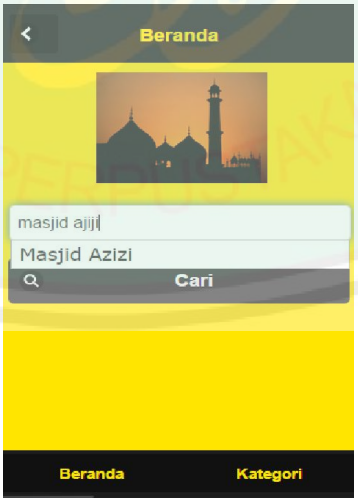
Gambar 4.20 Potongan *Source Code* Variabel Url, Query, QueryUrl dan AjaxResult

Setelah semua variabel dibuat, kemudian dilakukan *request* data *owl* dengan *AJAX*. **Gambar 4.21** menunjukkan proses *request* data *owl* menggunakan *method jquery AJAX*. Pada *method Ajax* ini, parameter *url* digunakan untuk menentukan *URL* dalam melakukan *request* data. Parameter *datatype* digunakan untuk menentukan tipe data ketika menampilkan respon dari *server*. Parameter *success* yaitu parameter yang dijalankan ketika *request* berhasil dilakukan. Dalam parameter *success* ini, respon *array* data *owl* masjid dimasukkan kedalam variabel AjaxResult yang telah diinisialisasi sebelumnya. Kemudian proses akhir yaitu dilakukan pemanggilan fungsi autokomplit sehingga dapat menampilkan saran *autocomplete*. Data yang digunakan dalam fungsi autokomplit yaitu *array* data

dari variabel ajaxResult. **Gambar 4.22** menunjukkan saran *autocomplete* dengan *levenshtein distance* ketika *user* salah mengetik ejaan *keyword*.

```
$.ajax(
    {
        type: 'POST',
        dataType: 'json',
        url: queryUrl,

        success: function(data)
        {
            var i;
            var len = data.results.bindings.length;
            var masjid;
            var masjid_name;
            var masjid_kab;
            var masjid_kec;

            for(i = 0; i < len; i++)
            {
              masjid = data.results.bindings[i];
              masjid_name = masjid.object.value;
              masjid_kab  = masjid.kabupaten.value;
              masjid_kec  = masjid.kecamatan.value;
              ajaxResult.push(masjid_name,masjid_kab,masjid_kec);
            }

             autokomplit();

        }
    });
```

Gambar 4.21 *Request Ajax* Untuk Menampilkan *Autocomplete*



Gambar 4.22 Saran *Autocomplete* dengan *Levenshtein distance*

4.3.2 *Request* Nama Masjid

Request nama masjid adalah proses untuk menampilkan nama masjid yang sesuai berdasarkan *keyword* pada kotak pencarian. *Keyword* tersebut dapat berupa nama masjid, nama kabupaten dan nama kecamatan. *Request* nama masjid akan diproses ketika *user* mengklik tombol cari pada aplikasi. Sebelum melakukan *request* data nama masjid, terlebih dahulu dibuat beberapa variabel yang dibutukan yaitu variabel term, variabel url, variabel query dan variabel queryUrl. Variabel term adalah variabel yang menampung nilai *input* pada kotak pencarian, variabel url untuk menampung *url* data, variabel query untuk menampung *query* dan variabel queryUrl untuk melakukan encode *query*. **Gambar 4.23** menunjukkan potongan *Source Code* dari variabel variabel term, variabel url, variabel query dan variabel queryUrl.

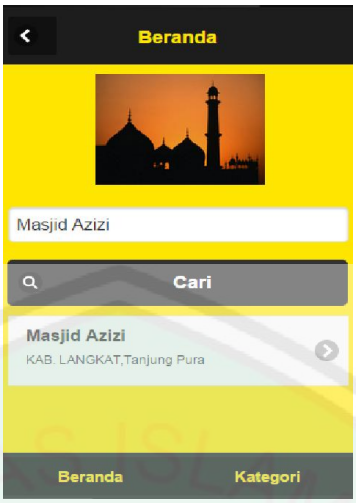
```
 var term = $("#searchid").val();
 var url = "http://127.0.0.1:3030/dataMasjidAsli/";

 var query = "PREFIX
 masjid:<http://simbi.kemenag.go.id/masjid#>
 SELECT ?object  WHERE
 {
   {
   ?subject masjid:nama  ?object.?subject masjid:kabupaten
?kab.    FILTER regex(?kab, '.*" + term + ".*', 'i')
   }
   UNION
   {
   ?subject masjid:nama  ?object.
   FILTER regex(?object, '.*" + term + ".*', 'i')
   }
   UNION
   {
   ?subject masjid:nama  ?object. ?subject masjid:kecamatan
?kec    FILTER regex(?kec, '.*" + term + ".*', 'i')
   }
 };
   var queryUrl = url + "?query="+encodeURIComponent(query);
```

Gambar 4.23 Potongan *Source Code* Variabel Term, Url, Query dan QueryUrl

Variabel query dari potongan *Source Code* di atas berfungsi untuk menampilkan hanya nama masjid berdasarkan *keyword* nama masjid, nama kabupaten, dan nama kecamatan. Setelah semua variabel yang dibutukan dibuat, kemudian dilakukan *request* data. **Gambar 4.24** menunjukkan *Source Code* untuk melakukan *request* nama masjid. Kemudian **Gambar 4.25** menunjukkan hasil pencarian nama masjid ketika *user* mengklik tombol cari.

```
$.ajax(
{
  dataType: "json",
  url: queryUrl,
  success: function( data )
  {
    var i;
    var len = data.results.bindings.length;
    var list;
    var masjid;
    var masjid_name;
    var result = $("#result6");
    result.empty();

    for(i = 0; i < len; i++)
    {
      masjid = data.results.bindings[i];
      masjid_name = masjid.object.value;
      list = '<li data-masjididcari="'+masjid_name+'"><a>';
      list += masjid_name;
      list += '</a></li>';
      $(result).append( $(list) );
    }
      $(result).listview('refresh');
  }
});
```

Gambar 4.24 *Request Ajax* untuk Menampilkan Hasil Pencarian



Gambar 4.25 Hasil Pencarian Nama Masjid

4.4 Pengujian

Pengujian pada aplikasi ini bertujuan mengetahui seberapa baik kualitas *Linked Open Data* yang digunakan dalam aplikasi dan juga seberapa tinggi kualitas hasil pencarian fitur *autocomplete* menggunakan metode *levenshtein distance*.

4.4.1 Hasil Pengujian Kualitas *Linked Open Data* (*LOD*)

Pengujian kualitas *Linked Open Data* dinilai berdasarkan skema bintang yang telah dibahas pada Bab II (Berners-Lee, 2006). Hasil pengujiannya dapat dilihat pada **Tabel 4.1** berikut:

Tabel 4.1 Hasil Pengujian Kualitas *LOD*

No	Kriteria	Butir Uji	Hasil Pengujian	Kesimpulan
1	Data tersedia pada *website* dengan lisensi terbuka / gratis dan bersifat *open data*.	Mengakses *Linked Open Data* melalui *Sparql Endpoint* (*URL*).	*Linked Open Data* dapat diakses melalui *URL* dengan gratis.	Satu Bintang

2	Tersedia sebagai data terstruktur yang dapat dipahami mesin.	Melakukan *query* menggunakan *Sparql Query* dengan berbagai properti / predikat terhadap *Linked Open Data*.	Dapat menampilkan data sesuai dengan predikat pada *query*	Dua Bintang
3	Kedua kriteria di atas ditambah dengan penggunaan format data terbuka / gratis (*JSON, XML* dll).	Melakukan *query* dengan berbagai properti / predikat terhadap *Linked Open Data* dalam berbagai format data terbuka.	Dapat menampilkan data dalam berbagai format data terbuka.	Tiga Bintang
4	Ketiga kriteria di atas ditambah dengan standar terbuka dari *W3C* (*RDF* dan *SPARQL*) untuk mengidentifikasi suatu subjek.	Melakukan *query* terhadap subjek *linked data* menggunakan berbagai properti / predikat pada *Sparql Query*.	Dapat menampilkan data menggunakan *Sparql Query* dengan berbagai predikat.	Empat Bintang
5	Semua kriteria di atas ditambah dengan menghubungkan data yang dimiliki dengan data dari sumber lain.	Melakukan pencarian data nama masjid pada aplikasi, jika data yang dicari sama dengan data Wikipedia maka akan menampilkan detil informasi dari Wikipedia.	*Linked Open Data* masjid tidak secara langsung terhubung dengan DBpedia, tetapi dapat dihubungkan melalui aplikasi.	Lima Bintang

Dari hasil pengujian kualitas *LOD* pada tabel di atas, bintang yang didapatkan yaitu dengan jumlah sempurna 5 bintang karena memenuhi semua kriteria pengujian kualitas *Linked Open Data*.

4.4.2 Hasil Pengujian Fitur *Autocomplete*

Pada tahap ini akan dilakukan pengujian metode *levenshtein distance* pada *autocomplete* untuk mengetahui seberapa tinggi kualitas hasil pencarian fitur *autocomplete* menggunakan metode *levenshtein distance*. Ukuran umum yang digunakan untuk mengukur kualitas hasil pencarian adalah kombinasi *precision* dan *recall* (Pamungkas Dkk, 2015). *Precision* adalah proporsi informasi relevan dari seluruh informasi yang berhasil ditemukan, sedangkan *Recall* adalah proporsi antara informasi relevan yang berhasil ditemukan dari seluruh informasi relevan yang ada di dalam aplikasi.

Untuk mendapatkan nilai *recall* dan *precision* tersebut perlu dicari terlebih dahulu nilai *True Positive (TP)* yaitu jumlah informasi relevan yang berhasil ditemukan, *False Positive (FP)* yaitu jumlah informasi yang berhasil ditemukan tetapi tidak relevan, dan *False Negative (FN)* yaitu jumlah informasi relevan tetapi tidak berhasil ditemukan. Nilai *precision (P)* didefinisikan sebagai, P = TP / (TP + FP). Sedangkan *recall (R)* didefinisikan sebagai, R = TP / (TP + FN). Nilai *recall* dan *precission* bernilai antara 0 s/d 1. Aplikasi diharapkan dapat memberikan hasil pencarian dengan nilai *precission* dan *recall* mendekati 1 (Salton, 1989).

Terdapat enam kemungkinan penyebab terjadinya kesalahan ejaan. Beberapa kesalahan ejaan yang mungkin terjadi pada penulisan beserta saran yang diberikan oleh aplikasi ditunjukkan pada **Tabel 4.2**.

Tabel 4.2 Kemungkinan Penyebab Kesalahan Ejaan

No	Kategori	Kata Salah	Kata Benar
1	Penggantian satu huruf	Kecamatan Kertasura	Kecamatan Kartasura
2	Penyisipan satu huruf	Masjid Rayya	Masjid Raya
3	Penghilangan satu huruf	Masjid Suhada	Masjid Syuhada
4	Penukaran dua huruf	Islamic Centre	Islamic Center
5	Penggantian dua huruf	Kecamatan Manonjoyo	Kecamatan Manonjaya
6	Kesalahan lebih dari dua huruf	Masjid Baitull Mutakin	Masjid Baitul Muttaqin

Pengujian dilakukan dengan menggunakan 30 inputan yang mewakili tiap kategori kesalahan ejaan. Hasil percobaan menggunakan *threshold* 80% ditampilkan pada **Tabel 4.3**.

Tabel 4.3 Hasil Pengujian *Levenshtein distance*

No	*Input*	Kategori Salah	*TP*	*FP*	*FN*	*Recall*	*Precision*
1	cupongkor	1	1	0	0	1	1
2	masjid kramas cijenuk	1	1	0	0	1	1
3	masjid singapuran	1	1	0	0	1	1
4	masjis nur	1	2	0	0	1	1

5	masjid fauzam	1	1	0	0	1	1
6	pelered	2	1	0	0	1	1
7	Masjid AL-FALLAH	2	3	7	0	1	0,33
8	Masjid thohharoh	2	1	0	0	1	1
9	Masjid baiturrohmann	2	6	10	0	1	0,38
10	Masjid darrus salam	2	3	3	0	1	0,5
11	Masjid an nihayah	3	1	0	0	1	1
12	kab. jmbang	3	1	2	0	1	0,33
13	kota makasar	3	1	0	0	1	1
14	Masjid HOZAINUL HIKMAH	3	1	2	0	1	0,33
15	Masjid sultonain	3	1	2	0	1	0,33
16	kedungwaur	4	1	0	0	1	1
17	masjid muhklisin	4	2	5	0	1	0,29
18	masjid silaturrahim	4	1	3	0	1	0,25
19	bangli	4	1	0	0	1	1
20	masjid raya sailangan	4	1	0	0	1	1
21	Masjid Raya Limay Purit	5	1	0	0	1	1
22	Masjid Sunan Kalijogo	5	1	0	0	1	1
23	sukomaknur	5	1	0	0	1	1
24	Masjid Sabilsl Muhtadim	5	1	2	0	1	0,33
25	kota salak	5	1	0	0	1	1

26	masjif tajkirr	6	1	0	0	1	1
27	KAB. KULOm PRiGs	6	1	0	0	1	1
28	Masjid BATUk IZAaH	6	1	0	0	1	1
29	kab. ngamjkk	6	1	0	0	1	1
30	kab. rrmbamgg	6	1	0	0	1	1

Dari hasil pengujian pada tabel di atas, didapatkan nilai rata-rata *precision* yaitu 0,8 dan nilai rata-rata *recall* yaitu 1. Dari hasil pengujian tersebut, aplikasi mampu memberikan saran *autocomplete* untuk tiap kategori kesalahan. Meskipun demikian, terdapat saran *autocomplete* yang tidak relevan dengan *keyword* yang diinput. Hal ini disebabkan oleh jumlah karakter yang mirip terlalu banyak. Walaupun terdapat saran *autocomplete* yang tidak relevan, namun saran *autocomplete* yang relevan tetap ditampilkan sehingga *user* tetap dapat memilih saran yang sesuai.

4.5 Integrasi Dalam Islam

Mengujungi masjid-masjid bersejarah selain bisa beribadah di dalamnya, juga sangat menarik untuk mempelajari sejarah atau arsitektur bangunan dari masjid tersebut. Karena dengan mempelajari sejarah, umat Islam bisa mengambil banyak pelajaran dari berbagai peristiwa masa lalu dan membenahi kesalahan guna memperoleh kejayaan dan kemuliaan di dunia dan akhirat.

Sebaik-baik kisah sejarah yang dapat diambil pelajaran darinya adalah kisah-kisah yang terdapat dalam ayat-ayat al-Qur'an dan hadits-hadits yang shahih dari Rasulullah Shallallahu 'alaihi wa sallam. Allah Subhanahu wa Ta'ala berfirman :

لَقَدْ كَانَ فِي قَصَصِهِمْ عِبْرَةٌ لِّأُولِي الْأَلْبَابِ ۗ مَا كَانَ حَدِيثًا يُفْتَرَىٰ وَلَٰكِن تَصْدِيقَ الَّذِي
بَيْنَ يَدَيْهِ وَتَفْصِيلَ كُلِّ شَيْءٍ وَهُدًى وَرَحْمَةً لِّقَوْمٍ يُؤْمِنُونَ ﴿١١١﴾

Artinya : 111. Sesungguhnya pada kisah-kisah mereka itu terdapat pengajaran bagi orang-orang yang mempunyai akal. Al Quran itu bukanlah cerita yang dibuat-buat, akan tetapi membenarkan (kitab-kitab) yang sebelumnya dan menjelaskan segala sesuatu, dan sebagai petunjuk dan rahmat bagi kaum yang beriman. (Qs. Yusuf/12: 111)

Dalam *Ibnu Katsir*, Allah berfirman bahwa sesungguhnya, dalam kisah para Rasul dan kaum mereka serta bagaimana Allah telah menyelamatkan orang-orang yang beriman dan menghancurkan orang-orang yang kafir: ‘ibratul li ulil albaabi maa kaana hadiitsaya yuftaraa (“Terdapat pengajaran bagi orang-yang berakal. Al-Qur’an itu bukanlah kitab yang dibuat-buat.”) maksudnya al-Qur’an itu tidak seharusnya di didustakan dan dibuat-buat dari selain Allah.

Wa laakin tashdiiqal ladzii baina yadaiHi (“Akan tetapi membenarkan kitab-kitab sebelumnya”) dari kitab-kitab yang diturunkan dari langit, dan membenarkan apa yang benar dari isinya, membantah pemutarbalikan, penyelewengan, dan perubahan yang terjadi di dalamnya, dan menentukan mana yang dinasakh (dihapus) atau ditetapkan.

Wa tafshiila kulla syai-in (“Dan menjelaskan segala sesuatu”) tentang halal, haram, sunnah, makruh, dan lain-lainnya. Seperti memerintahkan berbagai perbuatan taat, wajib, dan sunnah; dan melarang berbagai perbuatan haram dan sejenisnya, seperti makruh; memberitahukan hal-hal yang nyata dan ghaib yang akan datang, secara garis besar maupun rinci, memberitahukan tentang Rabb

Ta'ala, dengan nama-nama dan sifat-sifat-Nya dan ke-Mahasucian-Nya dari persamaan dengan makhluk-Nya.

Oleh karena itu, al-Qur'an adalah: Hudaw wa rahmatal li qaumiy yu'minuun ("Sebagai petunjuk dan rahmat bagi kaum yang beriman") yang membimbing hati mereka dari kesalahan menuju kebenaran, dari kesesatan menuju jalan yang lurus.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan implementasi dan pengujian aplikasi yang telah dilakukan, maka dapat ditarik kesimpulan sebagai berikut:

1. Hasil Uji Kualitas *Linked Open Data* menunjukkan hasil yang sangat baik dengan nilai bintang sempurna yaitu 5 bintang.
2. Hasil Uji *Precision* metode *Levenshtein Distance* pada Fitur *Autocomplete* dengan menggunakan *threshold* 80% menunjukkan saran *autocomplete* memiliki rata-rata *Precision* yang cukup baik yaitu 0,8. Tinggi rendahnya nilai *Precision* bergantung pada nilai *threshold* dan jumlah karakter yang mirip pada data. Semakin banyak jumlah karakter yang mirip pada data maka akan membuat *Precision* menjadi lebih rendah.
3. Hasil Uji *Recall* metode *Levenshtein Distance* pada Fitur *Autocomplete* dengan menggunakan *threshold* 80% menunjukkan saran *autocomplete* memiliki rata-rata *Recall* yang sempurna yaitu 1 karena sugesti *autocomplete* yang relevan selalu berhasil ditampilkan.

5.2 Saran

Saran dari hasil penelitian ini untuk pengembangan lebih lanjut yaitu dengan mendesain dan membangun file *OWL* yang terdapat *link* didalamnya untuk terhubung langsung dengan sumber data lain sehingga aplikasi dapat memperoleh data yang lebih luas dan dinamis secara langsung.

DAFTAR PUSTAKA

Andhika, Fatardhi Rizky. 2010. *Penerapan string suggestion dengan algoritma levenshtein distance dan alternatif algoritma lain dalam aplikasi*. Skripsi. Institut Teknologi Bandung.

Berners-Lee, Tim. 1998. *Semantic Web Road map*. Diakses dari https://www.w3.org/DesignIssues/Semantic.html. Diakses pada 29 Agustus 2015.

Berners-Lee, Tim. 2006. *Linked Data Design Issues*. Diakses dari https://www.w3.org/DesignIssues/LinkedData.html. Diakses pada 29 Agustus 2015.

Davies, Tim. 2010. *Linked Open Data Stack*. Diakses dari http://www.opendataimpacts.net/2011/05/whats-in-the-linked-open-data-stack/ . Diakses pada 16 September 2015.

Enrique Puertas, Maria Lorena Prieto dan Manuel de Buenaga. 2013. *Mobile Application For Accessing Biomedical Information Using Linked Open Data.*

Faruq, Umar. 2015. *Implementasi Damerau Levenshtein Distance dalam Pencarian Skripsi Berbasis Semantic Web pada Digital Library Universitas Islam Negeri Maulana Malik Ibrahim Malang*. Skripsi. Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Kusuma, Christian Sri, Achmad Ridok dan Indriati. 2013. *Perancangan Sistem Deteksi Plagiarisme Dokumen Teks Menggunakan Algoritma Damerau Levenshtein Distance.* Repositori Jurnal Mahasiswa PTIIK UB.

Kusuma, Muhammad Wachid. 2012. *Pencocokan string dalam fitur autocompletion pada text editor atau integrated development environment (IDE)*. Skripsi. Institut Teknologi Bandung.

Levenshtein, Vladimir I. (February 1966). *Binary codes capable of correcting deletions, insertions, and reversals.* Soviet Physics Doklady 10 (8): 707-710.

Morville, P. & Callender, Jeffery. 2010. *Search Patterns : Design for Discovery*. O'Reilly Media: Canada.

Pamungkas, Zanwar Yoga, Indriati dan Achmad Ridok. 2015. *Query Expansion pada Sistem Temu Kembali Informasi Dokumen Berbahasa Indonesia menggunakan Pseudo Relevance Feedback.* Repositori Jurnal Mahasiswa PTIIK UB.

Primadani, Yuli. 2014. *Simulasi Algoritma Levenshtein Distance untuk Fitur Autocomplete pada Aplikasi Katalog Perpustakaan*. Skripsi Universitas Sumatera Utara Medan.

Soeren Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak dan Zachary Ives. 2007. *DBpedia: A Nucleus for a Web of Open Data.* Springer.

Salton, G., 1989*, Automatic Text Processing, The Transformation, Analysis, and Retrieval of information by computer*. Addison-Wesly Publishing Company, Inc. USA.

Suryawan, Bernard Denata. 2013. *Pencarian Objek Wisata Berbasis Semantik*. Skripsi. Institut Teknologi Sepuluh Nopember Surabaya.

Taleski, Martin. 2014. *jQuery Autocomplete with spellcheck.* Diakses dari http://www.eyetea.mk/blog/2. Diakses pada 5 September 2015.

Weiss, Christian. 2013. *Transferring Open Government Data into the global Linked Open Data Cloud.*

Winoto, Hendri. 2012. *Deteksi Kemiripan Isi Dokumen Teks Menggunakan Algoritma Levenshtein Distance*. Skripsi. Universitas Islam Negeri Maulana Malik Ibrahim Malang.

Zayn, Afta Ramadhan. 2015. *Implementasi Ekstraksi Dokumen Web Masjid Bersejarah Untuk Membangun RDF Generator*. Skripsi. Universitas Islam Negeri Maulana Malik Ibrahim Malang.