**2. TINJAUAN PUSTAKA**

**2.1 Mesin Pencari (*Search Engine)***

Menurut Eko Supriyadi (2009) dalam bukunya “Teknologi Informasi dan Komunikasi Kelas XI” , mesin pencari (*search engine*) adalah salah satu program komputer yang di rancang khusus untuk membantu seseorang menemukan file-file yang disimpan dalam komputer, misalnya dalam sebuah web server umum di web (WWW) atau komputer sendiri. Mesin pencari memungkinkan kita untuk meminta *content* media dengan kriteria yang biasanya berisikan frase atau kata yang kita inginkan dan memperoleh daftar *file* yang memenuhi kriteria tersebut.

Untuk memudahkan pencarian di database yang begitu besar, mesin pencari menggunakan indeks untuk memilah-milah informasi yang ada di database. Sedangkan untuk memudahkan dan mempercepat pencarian, mesin pencari mempunyai metode pencarian tertentu yang sering di sebut algoritma pencarian, kecepatan dan ketepatan pencarian sebuah mesin pencarian tergantung kepada cara pengindeksan dan algoritma pencarian yang digunakan.

Adapun struktur umum sebuah mesin pencari adalah sebagai berikut :

1. Kotak teks pencari

Kotak ini digunakan sebagai tempat memasukan kata kunci yang akan dijadikan acuan dilakukan pencarian.

1. Tombol pencari

Tombol ini yang akan menjalankan perintah pencarian.

**2.1.1 Jenis-Jenis Mesin Pencari (*Search Engine*)**

Menurut Adli Abdillah Nababan (Nababan, Adli Abdillah, 2015) dalam skripsinya mengelompokan mesin pencari menjadi empat kategori berdasarkan cara mengumpulkan data halaman-halaman *website*, yaitu :

1. *Human Organized Search Engine*

Mesin pencari yang di kelola sepenuhnya oleh tangan manusia. Mesin pencari ini menggunakan metode dengan memilah-milih informasi yang relevan dan dikelompokan sedemikian rupa sehingga lebih bermakna dan bermanfaaat bagi penggunannya. Situs ini dalam prakteknya memperkerjakan para pakar dalam bidang-bidang tertentu, kemudian para pakar tersebut dapat mengkelompokan situs-situs tertentu sesuai dengan bidangnnya atau kategori situs itu sendiri.

1. *Computer Created Search Engine*

Mesin pencari kategori ini banyak memiliki kelebihan karena banyak menyajikan informasi walaupun kadang-kadang ada beberapa informasi yang tida relevan tidak seperti yang kita ingin kan. Mesin pencari ini telah menggunakan *spider software* yang berfungsi menyusup pada situs-situs tertentu, kemudian mengunpulkan data serta mengelompokan dengan sedikit bantuan tangan manusia.

1. *Hybrid Seacrh Engine*

Merupakan mesin pencari gabungan antara tangan manusia dengan komputer, sehingga menghasilkan hasil pencarian yang relatif akurat. Peran manusia dalam hal ini adalah sebagai penelaah dalam proses pengkoleksian *database* halaman *website*. Sebenarnya tipe ini lah yanag paling mudah pembuatannya karena dapat di desain sesuai dengan keinginan kita.

1. *MetaCrawler / Meta Search*

Merupakan perantara dari mesin pencari yang sebenarnya. Mesin ini hanya akan mengirimkan permintaan pencarian ke berbagai mesin pencari serta menampilkan hasilnya satu di layer browser sehingga akan menampilkan banyak sekali hasil dari berbagai mesin pencari yang ada.

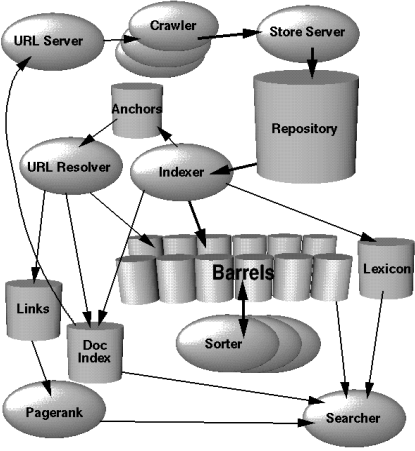
**2.1.2 Sistem Arsitektur Mesin Pencari (*Search Engine*)**

Sebelum mesin pencari dapat memberi tahu anda dimana *file* atau dokumen berada, *file* atau dokumen itu harus ditemukan terlebih dahulu. Menurut Monica Peshave dalam penulisannya berjudul “*How Serach Engines Work and A Web Crawer Application*” (2005), untuk mendapatkan informasi dalam ratusan juta halaman *website* yang tersedia, biasanya mesin pencari menggunakan piranti lunak khusus seperti robot dengan nama *spider*, untuk membuat daftar kata-kata yang ditemukan dalan *website*. Ketika *spider* membuat daftar, proses ini disebut sebagai *web crawling*. Sebuah program *web crawler* yang secara otomatis menjelajah *website* dengan mengunduh dokumen dan diikuti *link* dari halaman satu ke halaman lain. Keduanya biasanya dugunakan mesin pencari *website* untuk mengumpulkan data untuk indeksasi (*indexing*). Aplikasi lainnya yang ikut terlibat seperti validasi halaman, analisis terstruktur dan visualisasi, pemberitahuan pembaharuan, *mirroring* dan asisten/agen personal *website*, dan lain sebagainya. *Web crawler* juga biasa dikenal sebagai *spider, robot, worms,* dan lainya. *Crawler* merupakan program otomatis yang diikuti dengan *link* yang ditemukan pada halaman *website* (Nababan, Adli Abdillah, 2015).

Ada sebuah URL *Server* yang mengirimkan daftar URL-URL untuk diambil oleh *crawler*. Halaman *website* yang diambil kemudian dikirim ke *server* penyimpanan. *Server* penyimpanan tersebut kemudian memadatkan dan menyimpan halaman-halaman *website* menjadi sebuah repositori, lalu mengekstrak dokumen-dokumen, dan menguraikannya. Tiap dokumen diubah menjadi sebuah himpunan kata-kata yang terjadi disebut *hits*. *Hits* merekam kata-kata, posisi dokumen, dan ketepatan ukuran huruf serta penulisan huruf kapital. *Indexer* kemudian mendistribusikan *hits* menjadisuatu himpunan “*barrel*”, menciptakan indeks dengan pengurutan sebagian. *Indexer* berperan pada fungsi penting lainnya. *Indexer* mengurai seluruh *link* di tiap halaman *website* dan menyimpan informasi penting tentang halaman ke dalam *anchor file*. *File* ini berisi informasi yang cukup untuk menetapkan dimana tujuan *link* dan teks dari *link* (Nababan, Adli Abdillah, 2015).

*URL Resolver* membaca *anchor file* dan mengubah URL yang relatif menjadi URL yang mutlak dan dikembalikan menjadi ID dokumen. ID dokumen meletakan *anchor text* menjadi *forward index*, berkaitan dengan ID dokumen yang ditunjukan *anchor*. ID dokumen juga menyalakan *link database*, dimana terdapat beberapa pasangan ID dokumen. *Link database* digunakan untuk menghitung *Page Ranks* dari seluruh dokumen. (Nababan, Adli Abdillah, 2015)

*Sorter* mengambil seluruh *barrel*, dimana telah diurutkan berdasarkan ID dokumen dan diurutkan kembali berdasarkan ID kata untuk menghasilkan indeks yang terbalik. Hal ini telah dilakukan ditempat sehingga ruang memori temporer yang digunakan sedikit untuk menjalankan operasi ini. *Sorter* juga menghasilkan sebuah daftar berdasarkan ID kata-kata dan diubah menjadi indeks yang terbalik. Program dengan nama *Dump Lexicon* mengambil daftar tersebut bersamaan dengan leksikon yang dihasilkan olek *indexer* dan menghasilkan leksikon baru yang akan digunakan oleh pencari. Sebuah leksikon mendaftar semua istilahyang terjadi di indeks bersama dengan beberapa *term-level* *statistic* (misalnya jumlah keseluruhan dokumen dimana istilah terlibat) yang digunakan oleh algoritma pemeringkatan. Pencari dijalankan oleh *server website* dan digunakan untuk pembangunan leksikon oleh *Dump Lexicon* secara bersamaan dengan *inverted index* dan *Page Ranks* untuk menjawab *query* (Brin dan Page : 1998)*.*



**Gambar 2.1 Arsitektur Mesin Pencari Level Tinggi (Nababan, Adli Abdillah, 2015)**

**2.3 *Autocomplete***

Dikutip dari penelitian yang dilakukan David Ward, Jim Hahn, dan Kirsten Feist (Ward et al., 2012), *Autocompletion*, salah satu fitur yang menawarkan saran untuk pencarian yang ditampilkan saat pengguna mengetik teks dalam *search box*, telah diterapkan dimana-mana, baik di mesin pencari berskala besar maupun mesin pencari yang kecil pada situs individual. Pertama kali diperkenalkan sebagai fitur “*Google Suggest”* pada tahun 2004, *autocomplete* telah membuat terobosan ke ranah *library* melalui penyertaan dalam antarmuka dalam pencarian, termasuk paling terbaru antarmuka ProQuest dan dalam produk EBSCO. Penelitian tersebut juga menyimpulkan bahwa implementasi fungsi *autocomplete* yang menyumbang preferensi pengguna untuk membuat pengalaman pencarian yang menarik.

**2.4 *Autocorrect***

Kesalahan pengejaan kata sudah menjadi hal umum dalam query mesin pencari. Menurut Cucerzan dan Brill (Cucerzan & Brill, 2004), lebih dari 10% dari query mesin pencari diketik dengan salah. Ini bahkan lebih parah untuk *tail query*, dimana lebih dari 20% yang salah eja (Broder et al., 2009). Kesalahan pengejaan terjadi karena berbagai alasan. Saat mengetik cepat, pengguna dapat menambah atau menekan huruf tanpa sengaja. Selain salah ketik, beberapa kesalahan didapat dari kesulitan pengejaan kata itu sendiri.

Untuk membantu pengguna mengekspresikan informasi yang dibutuhkan, penting untuk mesin pencari secara otomatis mengoreksi kesalahan ejaan (Duan et al., 2011). Dua teknik secara umum dapat digunakan. Teknik pertama mengoreksi *query* setelah itu diserahkan ke mesin pencari. Untuk lebih meyakinkan, mesin pencari dapat mencari *query* yang telah dikoreksi secara langsung. Teknik yang kedua memberikan koreksi terhadap saran pelengkapan *query* sebagai *query* yang telah dimasukan. Secara khusus, mesin pencari mereespon setiap *query* yang terbaik dan benar serta menyelesaikan permintaan parsial.

**2.5 *Autosuggestion***

Pemberian saran pada *query* sangat diperlukan untuk membantu pengguna merumuskan *query*. Sebenarnya pemberian saran telah dianggap sebagai fitur yang harus dimiliki mesin pencari. Ada 2 pendekatan pemberian saran berdasarkan data yang digunakan, antara lain (Yang et al., 2008):

1. *Search Result Context*

Dibandingkan menggunakan keseluruhan isi halaman *website*, dengan hanya menggunakan judul dan potongan dari halaman *website* dari hasil pencarian teratas dapat digunakan untuk menghitung kata kunci yang diberikan. Suatu *query* dapat direpresenasikan sebagai



Dimana *rsr* mewakili relevansi antara *q*idengan frase ke-*j* dan *M* merupakan jumlah dari kemungkinan frase yang diberikan pada bahasa tertentu, misalnya Bahasa Inggris :



Mengingat beberapa *query* yang tidak populer kemungkinan hanya memberikan hasil yang sedikit, normalisasi frekuensi dari penggunaan kata, dimana *pj*  adalah sebuah kata, *Di,j*  adalah jumlah kejadian dari kata *pj* dalah sebuah himpunan judul dan potongan dari *query qi.* Menyerupai teknik pembobotan *TF-IDF,* digunakan perkalian frekuensi frase dan permintaan frekuensi invers untuk menurunkan bobot frase. *Q* merupakan himpunan dari *query*, sedangkan *Sri* merupakan hasil pencarian kontent terhadap *query qi.*

1. *Query Log Session Context*

*Query* terjadi dengan permintaan pengguna yang disampaikan dalam sejumlah sesi *log query* dapat digunakan sebagai frase kunci. *Query* dapat berikatan jika *query* tersebut muncul dalam sejumlah besar sesi pengguna *log query* (*query* berturut-turut). Algoritma *log session-based* memanfaatkan pengetahuan tentang catatan penggunaan permintaan dari beberapa penguna. Karena itu, *query* yang bermanfaat, dimana tidak mengandung string asli dari *query*, dapat diperikan saran (*suggestion*)*. Query qi* dapat direpresentasikan sebagai berikut :

**

Dimana *M* merupakan jumlah kemungkinan frase. Kesamaan fungsi pembobotan tersebut dapat digambarkan sebagai berikut :



Dimana *pj* merupakan frase, *Ni,j* adalah jumlah kemungkinan frase *pj* dalam sesi yang digunakan *query qi, Q* adalah himpunan *query*, dan *logi* merupakan *log session* konten yang berkaitan dengan *query qi.*

**Gambar 2.2**  **Cuplikasn Pemberian *Suggestion* dalam Sistem (Yang et al., 2008)**

Untuk *query* yang baru diberikan yang digambarkan dengan simbol *qsub,* kita juga dapat menggambarkan *query* tersebut menggunakan kedua metode pemberian *suggestion* yang telah dijelaskan sebelumnya dan nilai relevansi terhadap *query* yang disarankan (*qsug*). Menyerupai mesin pencari, baik pemberian peringkat secara dinamis maupun pemberian peringkat secara statis digambungkan sebagai pencarian *website*.

Pengguna mungkin tidak mengetahui hubungan antara *query* dan *suggestion* yang diberikan. Dengan menjelaskan hubungan antara *query* yang disarankan dengan *query* yang dimasukan kemungkinan akan membantu pengguna untuk merumuskan *query* yang selanjutnya.

Berdasarkan ketepatan srategi untuk *query* dalam mesin pencari, terdapat dua jenis *query* yang bersamaan digunakan untuk mesin pencari. Penggabungan *query* tersebut disimbolkan dengan kesamaan *qjoint* (menggabungkan dua *query* dengan karakter kosong). Relevansi terhadap potongan *query* dapat digambarkan sebagai berikut :



**2.6 *Natural Language Processing* (NLP)**

Bahasa adalah suatu sistem untuk komunikasi yang terdiri atas simbol dan aturan yang digunakan untuk mengekspresikan ide, pikiran dan perasaan. Menurut Hadyan Taris Akbar (2013 : 6), *Natural Language Processing* (NLP) atau pemrosesan bahasa alami sebagai suatu bidang ilmu *Artificial Intelligence* (AI), mempelajari komunikasi antara manusia dengan komputer untuk memahami arti dari teks/tulisan yang diberikan dalam format bahasa alami dan memberikan respon yang sesuai, kisalnya dengan melakukan suatu aksi tertentu atau menampilkan data tertentu. Sebuah *Natural Language System* harus memperhatikan pengetahuan terhadap bahasa itu sendiri, baik dari segi kata yang digunakan, bagaimana kata-kata tersebut digabung untuk menghasilkan suatu kalimat, apa arti sebuah kata, apa fungsi sebuah kata dalam sebuah kalimat dan sebagainya. Bahasa dapat dibedakan menjadi dua, yaitu Bahasa Alami dan Bahasa Buatan. Bahasa Alami adalah bahasa yang biasa digunakan untuk berkomunikasi antar manusia, sedangkan Bahasa Buatan adalah bahasa yang dibuat secara khusus untuk memenuhi kebutuhan tertentu, misalnya bahasa pemrograman komputer.

**2.6.1 Tahap Analisis NLP**

Menurut Desiani et al. (Desiani et al., 2006), secara umum ada tiga tahap untuk analisis pemahaman bahasa alami, yaitu :

1. *Parsing* (penguraian) merupakan tindakan mengurai struktur sintaks kalimat. *Parsing* tidak hanya membuktikan bahwa kalimat dapat dibentuk secara sintaks, tetapi juga dapat menentukan struktur linguistiknya. Dengan mengidentifikasi hubungan linguistik utamanya seperti *subjek-verb, verb-objek,*  dan *noun-modifier*, maka penguraian (*parser*) memberikan suatu kerangka/bentuk diagram bagi interpretasi semantik dan hal ini sering kali digambarkan dalam pohon pengurai (*parse tree*). Penguraian menggunakan pengetahuan sintaks bahasa, morfologi, dan beberapa semantik bahasa.
2. Interpretasi semantik, pada tahap ini menghasilkan suatu representasi arti dari teks. Representasi lain umumnya digunakan, meliputi ketergantungan konseptual, bingkai dan representasi berbasis logika. Interpretasi semantik menggunakan pengetahuan tentang arti kata dan struktur linguistiknya, seperti peranan kata benda atau transitivitas kata kerja. Tahap ini melakukan pengujian konsistensi semantik.
3. Struktur-struktur dari basis pengetahuan ditambahkan pada representasi kalimat untuk menghasilkan representasi perluasan dari arti kalimat. Struktur yangdihasilkan merepresentasikan makna dari teks bahasa alami dan digunakan oleh sistem untuk pemrosesan lebih jauh.

**2.6.2 Pemrosesan NLP**

Pemrosesan bahasa alami dapat dibagi atas beberapa proses (Akbar, H.T., 2013), yaitu:

1. ***Morphological analysis****,* dimana kata secara individu dianalisis ke dalam komponen-komponennya dan yang bukan kata dipisahkan dari kata.
2. ***Syntatic analysis,*** urutan linier dari kata ditransformasikan ke dalam struktur yang menunjukan bagaimana kata saling berhubungan. Urutan kata yang melanggar aturan bahasa dalam pengkombinasian kata akan ditolak.
3. ***Semantic analysis***, dibuat *mapping* antara struktur sintaksis dan objek dalam domain kerja. Struktur dimana tidak ada *mapping* yang memungkinkan, akan ditolak.
4. ***Discourse integration***, diana arti dari kalimat secara individu bergantung pada kalimat-kalimat yang mendahului dan mungkin dapat mempengaruhi arti dari kalimat-kalimat yang mengikutinya.
5. ***Pragmatic analysis***, dimana struktur yang merepresentasikan apa yang diucapkan, diinterpretasikan kembali untuk menentukan apa arti sebenarnya.

**2.6.3 *N-Gram***

*N-gram* adalah potongan n karakter dalam suatu string tertentu atau potongan n kata dalam suatu kalimat tertentu (Manalu, Utomo 2014). Misalnya dalam kata “Teknik” akan didapatkan n-gram sebagai berikut.

**Tabel 2.1 Contoh Pemotongan N-Gram Berbasis Karakter**

|  |  |
| --- | --- |
| **Nama** | **N-gram Karakter** |
| Uni-gram | T, E, K, N, I, K |
| Bi-gram | \_T, TE, EK, KN, NI, IK, K\_ |
| Tri-gram | \_TE, TEK, EKN, KNI, NIK, IK\_, K\_\_ |
| Quad-gram | \_TEK, TEKN, EKNI, KNIK, NIK\_, IK\_\_, K\_\_\_ |

Karakter blank “\_” digunakan untuk merepresentasikan spasi di depan dan diakhir kata. Dan untuk *word-based* n-gram contohnya adalah sebagai berikut. Kalimat : “N-gram adalah potongan n karakter dalam suatu string tertentu”

**Tabel 2.2 Contoh Pemotongan N-Gram Berbasis Kata**

|  |  |
| --- | --- |
| **Nama** | **N-gram Kata** |
| Uni-gram | n-gram, adalah, potongan, n, karakter, dalam, suatu, sring, tertentu |
| Bi-gram | n-gram adalah, adalah potongan, potongan n, n karakter, karakter dalam, dalam suatu, suatu string, string tertentu |
| Tri-gram | n-gram adalah potongan, adalah potongan n, potongan n karakter, n karakter dalam, karakter dalam suatu, dalam suatu string, suatu string tertentu |

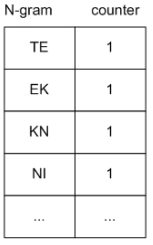
**2.6.3.1 *N-Gram Based Text Categorization***

Bahasa manusia memiliki beberapa kata yang muncul (digunakan) lebih sering dibandingkan dengan kata yang lain (Manalu, Utomo 2014).

*1. Learning*

Setelah dilakukan *preprocessing* terhadap dokumen-dokumen dalam *training set*, maka selanjutnya dilakukan *learning* terhadap dokumen-dokumen tersebut. Langkah-langkah *learning* yang dilakukan adalah sebagai berikut :

* Fitur-fitur (*token*) yang telah didapatkan ditransformasikan ke dalam bentuk ngram dengan n = 2, 3, dan 4.
* Masukkan tiap-tiap *n-gram* yang telah didapatkan dalam suatu tabel hash sebagai counter untuk menghitung frekuensi *n-gram* dalam dokumen. Tabel hash tersebut menggunakan mekanisme penanganan duplikasi konvensional untuk menjamin bahwa setiap *n-gram* memiliki *counter*-nya masing-masing. Contoh implementasi dari mekanisme ini dijelaskan dalam gambar di bawah ini.



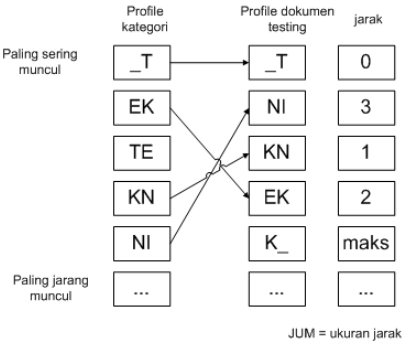
**Gambar 2.3 Contoh Penggunaan Tabel *Hash* (Manalu, 2014)**

Ketika muncul *n-gram* “TE” lagi, maka frekuensi (*counter*) “TE” ditambah 1, tidak lagi ditambahkan baris baru dalam tabel *hash* tersebut. Sehingga duplikasi dapat dicegah. Setelah semuanya dihitung, keluarkan semua *N-gram* beserta jumlah kemunculannya. Urutkan *N-gram* dalam urutan terbalik berdasarkan jumlah kemunculannya.

Hasil akhir dari proses diatas adalah *N-gram frequency profile* dari dokumen. Setelah didapatkan *N-gram frequency profile* dari dokumen (per kategori dalam *training set*), untuk *testing*-nya maka dilakukan pengukuran jarak profil kategori dengan profil dokumen yang akan diketahui kategorinya.

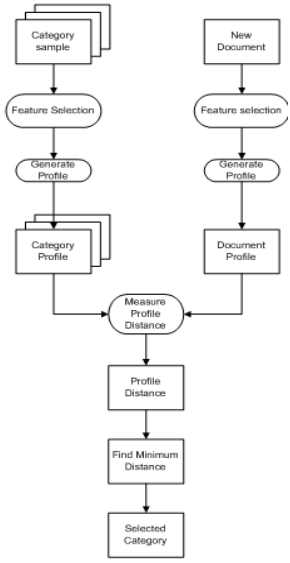
2. *Testing*

Seperti yang telah dijelaskan dalam sebelumnya, untuk melakukan testing terhadap sebuah dokumen, maka dilakukan langkah-langkah seperti pada proses *training* terhadap dokumen dalam *test-set*. Dengan demikian, didapatkan *N-gram* *frequency profile* untuk dokumen *testing*. Kemudian langkah yang selanjutnya untuk mengetahui kategori dari dokumen *testing* adalah dengan menghitung jarak antara profil dokumen testing dengan profil dari masing-masing kategori dalam dokumen *training*. Pengukuran jarak (*distance measure*) dilakukan dengan mekanisme *out-of-place measure*. Cara kerja mekanisme ini adalah sebagai berikut. Untuk setiap *N-gram* dalam profil dalam dokumen *testing*, temukan profil yang sama pada profil kategori dalam dokumen *training*. Kemudian hitung seberapa jauh profil tersebut dari tempat yang seharusnya jika dokumen tersebut termasuk dalam suatu kategori. Untuk lebih jelasnya, dapat dilihat dalam gambar dibawah ini.



**Gambar 2.4 Contoh Penghitungan Jarak Dengan Mekanisme *Out-Of-Place Measure* (Manalu, 2014)**

*N-gram* yang muncul dalam dokumen *testing* namun tidak muncul dalam profil kategori diberi jarak maksimal yaitu jumlah keseluruhan *N-gram* yang terbentuk. Kategori dari dokumen *testing* tersebut merupakan kategori dengan ukuran jarak (*distance measure*) terkecil. Sebagai catatan, profil diatas hanya untuk menjelaskan saja, dan bukan refleksi dari *N-gram frequency statistic* yang sebenarnya. Proses kategorisasi teks secara umum dapat dilihat pada gambar di bawah ini (Manalu, Utomo 2014).



**Gambar 2.5 Gambaran Umum Kategorisasi Teks Dengan Menggunakan *N-gram* (Manalu, Utomo 2014)**

**2.6.4 *Stemming* dan *Lemmatization***

Untuk alasan tata bahasa, dokumen akan menggunakan berbagai bentuk kata, seperti mengatur, menyelenggarakan, dan pengorganisasian (Manning et al., 2008). Selain itu, ada kumpulan dari kata-kata yang berhubungan secara derivatif dengan arti yang sama, seperti demokrasi, demokratik, dan demokratisasi. Dalam bayak situasi, tampaknya seolah-olah hal ini akan berguna untuk pencarian salah satu kata dari kata-kata lain untuk mengembalikan dokumen yang mengandung kata lain dalam satu himpunan.

Tujan dari *stemming* dan *lemmatization* adalah untuk mengurangi bentuk inflektif dan bentuk kadang-kadang terkait secara derivatif dari kata ke bentuk dasar kata umum. Contohnya :

am*, are, is ® be  
 car, cars, car's, cars' ® car*

Hasil dari pemetaan teks akan menjadi sesuatu seperti berikut :

*The boy's cars are different colors* ® *The boy car be differ color*

Namun, terdapat dua kata yang berbeda rasa. *Stemming* biasanya mengacu pada proses heuristik mentah yang memotong ujung kata-kata dengan harapan mencapai tujuan dengan tepat, dan sering termasuk penghapusan afiks derivatif. *Lemmatization* biasanya mangacu pada melakukan hal-hal benar dengan penggunaan kosakata dan analisis morfologi kata-kata, biasanya bertujuan untuk menghilangkan ujung infleksi saja dan untuk mengembalikan basis atau bentuk kamus kata, yang dikenal dengan *lemma*. Jika dihadapkan dengan melihat tanda, *stemming* mungkin hanya mengembalikan huruf 's', sedangkan *lemmatization* a akan berusaha untuk kembali baik hanya melihat atai memperhatikan tergantung pada apakahpenggunaan *token* sebagai kata kerja atau kata benda. Keduanya juga mungkin berbeda dalam membendung kata-kata yang yang paling sering runtuh terkait secara derivatif, sedangkan *lemmatization* pada umumnya hanya meruntuhkan infleksi yang berbeda bentuk dari *lemma.* Pengolahan linguistik untuk *stemming* atau *lemmatization* sering dilakukan oleh komponen *plug-in* tambahan untuk proses pengindeksan, dan sejumlah komponen seperti itu ada, baik secara komersial maupun *open source* (Manning et al., 2008).

Algoritma yang paling umum untuk *stemming* bahasa Inggris dan salah satun yang telah berulang kali terbukti secara empiris sangatefektif adalh algoritma Porter. Algoritma Porter terdiri dari 5 fase pengurangan kata, diterapkan secara berurutan. Dalam tiap tahap ada berbagai konvensi untuk memilih aturan, seperti memilih aturan dari masing-masing kelompok aturan yang berlaku untuk akhiran terpanjang. Pada tahap pertama, konvensi ini digunakan dengan kelompok aturan sebagai berikut (Manning et al., 2008):

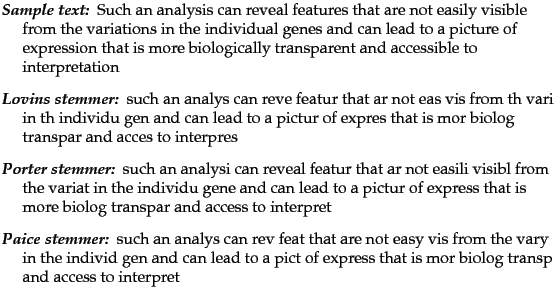
**Tabel 2.3 Contoh Tahap** **Pertama Algoritma Porter**

|  |  |
| --- | --- |
| **Aturan** | **Contoh** |
| SSES ® SS  IES ® I  SS ® SS  S ® | caresses ® caress  ponies ® poni  caress ® caress  cats ® cat |

Banyak aturan kemudian menggunakan konsep ukuran dari sebuah kata, yang secara longgar memerikasa jumlah suku kata untuk melihat apakah sebuah kata yang cukup panjang yang masuk akal untuk menganggap bagian pencocokan aturan sebagai akhiran bukan sebagai bagian dari batang dari sebuah kata (Manning et al., 2008). Misalnya, aturan :

(*m* > 1) EMENT ®

Akan memetakan pengganti untuk '*replac'*, tapi bukan *'cement'* ke '*c'*. Terdapat *sttemer* lainnya, termasuk yang tertua, yaitu *stemmer* Lovins, dan yang terbaru *stemmer* Paice/Husk (Manning et al., 2008). Berikut ini perbandingan algoritma *stemming* :



**Gambar 2.6 Perbandingan Algoritma *Stemming* (Manning et. al., 2008)**

Gambar 2.6 menyajikan pebandingan informal tentang perilaku yang berbeda dari beberapa *stemmer*. *Stemmer* menggunakan aturan bahasa tertentu, tetapi mereka memerlukan pengetahuan yang kurang dari *lemmatizer*, yang membutuhkan kosakata yang lengkap dan analisis morfologi dengan benar *lemmatize* kata. Domain tertentu mungkin juga memerlukan aturan yang khusus. Namun, *stemmed* dalam bentuk yang sebenarnya tidak masalah, hanya kelas ekivalen yang membentuknya.

Dibandingkan menggunakan *stemmer*, *lemmatizer* dapat digunakan, yang merupakan salah satu alat *Natural Language Processing* (NLP) yang melakukan analisis morfologi penuh untuk mengidentifikasi *lemma* secara akurat untuk setiap kata. Melakukan analisis morfologi penuh menghasilkan paling banyak manfaat yang sangat sederhana untuk pengambilan. Sementara itu banyak membantu untuk beberapa pertanyaan, yang yama menyakitkan banyak kinerja untuk orang lain. *Stemming* meningkat mengingat ketika dalam presisi yang merugikan. Sebagai contoh dari apa yang bisa salah, perhatikan *stemmer*  Porter telah melakukan *'stem'* terhadap semua kata berikut (Manning et al., 2008):

*operate operating operates operation operative operatives operational*

Namun, karena telah beroperasi di berbagai bentuk merupakan kata kerja yang umum, kita akan berharap untuk kehilangan presisi yang cukup besar pada permintaan seperti berikut menggunakan *stemmer* Porter :

*operational and research  
 operating and system  
 operative and dentistry*

Untuk kasus seperti ini, berpindah pilihan untuk menggunakan *lemmatizer* tidakakan secara lengkap menyelesaikan masalah, karena beberapa bentuk infleksi telah digunakan pada beberapa pemindahan tempat: kalimat dengan kata yang beroperasi dan sistem tidak cocok untuk menjalankan *query* dan sistem. Mendapatkan nilai yang lebih baik dari bentuk normalisasi lebih bergantung pada isu-isu pragmatis penggunaan kata dari pada isu-isu formal morfologi linguistik (Manning et al., 2008).

**2.6.5 *Tokenization***

Dengan adanya urutan karakter dan unit dokumen yang didefinisikan, *tokenization* merupakan suatu tugas yang memotong urutan karakter menjadi beberapa potongan, yang disebut sebagai *token* , yang kemungkinan di saat yang sama membuang karakter tertentu, seperti tanda baca (Manning et al., 2008). Berikut ini adalah contoh dari *tokenization*:



*Token* ini biasanya dengan bebas disebut sebagai *term* atau kata-kata. *Token* adalah sebuah contoh dari urutan karakter dalam beberapa dokumen tertentu yang dikelompokan bersama sebagai unit semantik yang berguna untuk pengolahan. *Term* adalah kelas dari semua *token* yang berisi urutan karakter yang sama. Sebuah *term* merupakan (kemungkinan dinormalisasi) jenis yang masuk dalam kamus sistem IR(*Information Retrieval*). Himpunan dari indeks *term* bisa sepenuhnya berbeda dari *token*, misalnya mereka bisa menjadi pengidentifikasi semantik dalam taksonomi, namun dalam prakteknya dalam sistem IR yang modern, mereka sangat terkait dengan *token* dalam dokumen. Namun, bukannya menjadi persis dengan *token* yang muncul dalam dokumen, mereka biasanya berasal dari mereka sendiri dengan berbagai proses normalisasi. Sebagai contoh, jika dokumen yang diindeks adalah '*to sleep perhance to dream*', maka da 5 token, tetapi hanya ada 4 tipe (karena ada 2 contoh). Namun jika untuk dihilangkan dari indeks (sebagai kata perhentian), maka terdapat hanya 3 *term*, yaitu: *sleep, perhance,* dan *dream* (Manning et al., 2008)*.*

**2.6.6****TF-IDF *Weighting***

Manning et. al. mengkombinasikan definisi dari *term frequency* dan *inverse document frequency*, untuk menghasilkan sebuah bobot campuran dari tiap *term* dari tiap dokumen (Manning et al., 2008). Skema TF-IDF *weighting* memberikan jangka t o berat di d dokumen yang diberikan oleh rumusan berikut ini

tf-idf*t,d*= tf*t,d* × idf

Dengan kata lain, tf-idf*t,d* menetapkan *term* *t* sebuah nilai dalam dokumen *d* yang memiliki sifat sebagai berikut :

1. Nilai tf-idf*t,d* akan tinggi saat *ter t* terjadi berulang kali dalam sejumlah kecil dokumen
2. Nilai tf-idf*t,d* akan rendah ketika *term t*  muncul lebih sedikit dalam dokumen, atau terjadi dalam banyak dokumen
3. Nilai tf-idf*t,d* terendah ketika term muncul di hampir semua dokumen

Pada titik ini, kita dapat melihat setiap dokumen sebagai vektor dengan satu komponen yang sesuai untuk setiap *term* dalam kamus, bersama-sama dengan bobot masing-masing komponen yang diberikan rumus sebelumnya. Untuk kamus *term* yang tidak terjadi dalam dokumen, bobot ini bernilai nol. Bentuk vektor ini akan dibuktikan menjadi lebih penting untuk pemberian bobot dan pemberian peringkat. Sebagai langkah awal, Manning et. al. memperkenalkan ukuran nilai yang tumpang tindih, dimana: skor dokumen *d* merupakan jumlahnya, jumlah dari seluruh *term*, dari jumlah perkalian setiap *query term* yang terlibat ada dokumen *d*. kita dapat memperbaiki ide ini sehingga kita menambahkan jumlahnya, bukan jumlah kejadian dari tiap *query term t* dalam dokumen *d*, namun bobot TF-IDFdari tiap *term* di dokumen (Manning et al., 2008).



**2.7 Logika *Fuzzy***

Logika *fuzzy* merupakan salah satu komponen pembentuk *soft computing*. Logika *fuzzy* pertama kali diperkenalkan oleh Prof. Lotfi A. Zadeh pada tahun 1965. Dasar logika *fuzzy* adalah teori himpunan *fuzzy*. Pada teori himpunan *fuzzy*, peranan derajat keanggotaan sebagai penentu keberadaan elemen dalam suatu himpunan sangatlah penting. Nilai keanggotaan atau derajat keanggotaan atau membership *function* menjadi ciri utama dari penalaran dengan logika *fuzzy* tersebut (Kusumadewi, 2010).

Logika *fuzzy* dapat dianggap sebagai kotak hitam yang menghubungkan antara ruang input menuju ruang output (Gelley, 2000, dari Kusumadewi, 2010). Kotak hitam tersebut berisi cara atau metode yang dapat digunakan untuk mengolah data input menjadi output dalam bentuk informasi yang baik.

Dikutip dari penulisan Sundari Retno Andani (Andani, Sundari Retno, 2012), ada beberapa alasan digunakannya logika fuzzy :

1. Konsep logika *fuzzy* mudah dimengerti. Karena logika *fuzzy* menggunakan dasar teori himpunan, maka konsep matematis yang mendasari penalaran *fuzzy* tersebut cukup mudah untuk dimengerti.
2. Logika *fuzzy* sangat fleksibel, artinya mampu beradaptasi dengan perubahan-perubahan, dan ketidakpastian yang menyertai permasalahan.
3. Logika *fuzzy* memiliki toleransi terhadap data yang tidak tepat. Jika diberikan sekelompok data yang cukup homogen, dan kemudian ada beberapa data yang “eksklusif”, maka logika *fuzzy* memiliki kemampuan untuk menangani data eksklusif tersebut.
4. Logika *fuzzy* mampu memodelkan fungsi-fungsi nonlinier yang sangat kompleks.
5. Logika *fuzzy* dapat membangun dan mengaplikasikan pengalamanpengalaman para pakar secara langsung tanpa harus melalui proses pelatihan. Dalam hal ini, sering dikenal dengan nama *fuzzy* *expert* system menjadi bagian terpenting.
6. Logika *fuzzy* dapat bekerjasama dengan teknik-teknik kendali secara konvensional. Hal ini umumnya terjadi pada aplikasi di bidang teknik mesin maupun teknik elektro.
7. Logika *fuzzy* didasarkan pada bahasa alami. Logika *fuzzy* menggunakan bahasa sehari-hari sehingga mudah dimengerti.

**2.8 *E-Commerce***

Pada bagian ini akan menjelaskan definisi, perbedaan, tipe dan faktor penentu utama *e-commerce* yang dikutip dari penulisan Wulang Saputra (2014 : 5).

**2.8.1 Definisi *E-Commerce***

*E-commerce* adalah aktivitas bisnis *online* yang jangkauannya luas untuk produk dan jasa. *E-commerce* juga berkenaan kepada segala bentuk transaksi bisnis dengan pengguna berinteraksi secara elektronik dibandingkan bertukaran secara fisik ataupun secara langsung (Khurana, Hitesh, et al., 2011). *E-commerce* adalah kegunaan dari komunikasi elektronik dan teknologi pemrosesan informasi digital di transaksi bisnis untuk membuat, merubah, dan mendefinisikan hubungan antar organisasi dan antar organisasi dengan individu.

**2.8.2 Perbedaan *E-Commerce dengan E-Business***

Menurut Hitesh Khurana et al (Khurana, Hitesh, et al., 2011), *E-commerce* dan *E-business* mempunyai konsep yang berbeda. Pada *e-commerce, information and communication* (ICT) digunakan pada transaksi antar bisnis atau antar organisasi (transaksi antar dan diantara perusahaan organisasi) dan pada transaksi *business-to-consumer* (transaksi antara perusahaan/organisasi dan individu).

*E-business* di sisi lain, ICT digunakan untuk meningkatkan bisnis seseorang. Hal ini mencakup proses apapun yang organisasi bisnis (baik *for-profit,* pemerintah, atau *non-pfrofit*) lakukan lewat *computer-mediated-network*. Definisi yang lebih komprefensif dari *e-business* adalah “Transformasi proses organisasi untuk memberikan nilai tambahan kepada *customer* melalui penerapan teknologi, filsafat, dan paradigma komputasi ekonomi baru” (Khurana, Hitesh, et al., 2011).

**2.8.3 Tipe *E-Commerce***

Terdapat tipe-tipe umum yang biasa digunakan (Khurana, Hitesh, et al., 2011), yaitu :

**2.8.3.1 *Business-to-business* (B2B)**

B2B *e-commerce*, singkatnya diartikan sebagai *e-commerce* antara perusahaan. Menurut Hitesh Khurana et al (Khurana, Hitesh, et al., 2011), sekitar 80% *e-commerce* adalah tipe ini, dan kebanyakan ahli memprediksikan bahwa B2B *e-commerce* akan berkembang dengan cepat dibandingkan dengan B2C.

Pasar B2B mempunyai 2 komponen utama *e-frastucture* dan *e-markets. E-frastucture* adalah arsitektur dari B2B, yang intinya meliputi:

1. Logistik = transportasi, *warehousing* dan distribusi (contoh: *Procter dan Gamble*)
2. Penyedia layanan aplikasi = penyebaran, hosting dan manajemen paket perangkat lunak dari fasilitas pusat (contoh : *Oracle, Link share*)
3. *Outsourcing* fungsi pada proses *e-commerce*, seperti *web hosting,* , keamanaan dan solusi layanan pelanggan (contoh : *e-share, Net Sales, iXL Enterprises*)
4. Perangkat lunak solusi lelang untuk operasi dan pemeliharaan lelang di internet (contoh : *Moai Technologies* dan *Open Site Technologies*).
5. Perangkat lunak manajemen konten untuk fasilitas manajemen konten situr *website* dan pengiriman (contoh : *Interwoven* dan *Procure Net*).
6. *Enabler* perdagangan berbasis web (contoh: *Commerce One, XMLenabled purchasing automation software*).

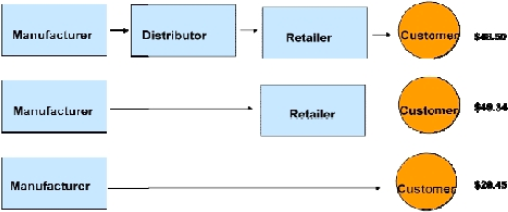
*E-market* secara sederhana dapat diartikan sebagai *website* dengan pembeli dan penjual berinteraksi satu dengan lainnya dan melakukan transaksi.

**2.8.3.2 *Business-to-consumer* (B2C)**

*Business to consumer e-commerce* atau perdagangan antar perusahaan dengan konsumen, meliputi konsumen mengumpulkan informasi, pembelian barang (seperti produk kosumen atau buku) atau barang berupa informasi (berupa konten digital) dan untuk barang berupa informasi, peneriamaan produk melalui jaringan elektronik (Khurana, Hitesh, et al., 2011).

B2C *e-commerce* mengurangi biaya transaksi dengan meningkatkan akses pelanggan untuk informasi dan memungkinkan pelanggan untuk menemukan harga yang paling bersaing untuk produk atau layanan.

Pada kasus barang informasi, B2C *e-commerce* bahkan lebih menarik karena menghemat perusahaan dari biaya tambahan jaringan distribusi. Selain itu, negara dengan pertumbuhan dan populai internet yang kuat, pengiriman barang berupa informasi menjadi semakin layak. Gambar 2.5 merupakan cara kerja dari tipe *e-commerce business to consumer*.

**

**Gambar 2.7 Cara Kerja B2C (Khurana, Hitesh, et al, 2011)**

**2.8.3.2 *Business-to-gonverment* (B2G)**

*Business-to-goverment e-commerce* umumnya diartikan sebagi perdagangan antara perusahaan dengan sektor publik. Hal ini mengacu kepada penggunaan internet untuk pengadaan publik, prosedur perizinan, dan operasi yang berhubungan dengan pemerintahan lainnya.

**2.8.3.3 *Consumer-to-consumer* (C2C)**

*Consumer-to-consumer e-commerce* atau C2C adalah perdagangan antara individu atau konsumen. Tipe perdagangan ini ditandai oleh perkembangan pasar elektronik dan lelang *online*, khususnya di industri dengan perusahaan/usaha dapat mengajukan tawaran untuk apa yang mereka mau dari beberapa *supplier.*

**2.8.3.1 *Mobile commerce (m-commerce)***

*Mobile commerce* adalah membeli atau menjual barang dan jasa melalui teknologi *wireless*,seperti *handphone* atau *personal digital assistans* (PDAs). Jepang dilihat sebagai pemimpin pada *m-commerce* ini.

Pengiriman konten melalui *wireless device* menjadi cepat dan lebih aman. Beberapa orang percaya bahwa *m-commerce* ini akan melampaui *wire line e-commerce* sebagai pilihan untuk transaksi perdagangan digital. Hal ini mungkin terjadi di kawasan Asia-Pasifik dengan penggunaan *mobile phone* lebih banyak dari pengguna internet.

**2.8.4 Faktor Penentu Utama *E-Commerce***

Menurut F.D. Davis, *Technology Acceptance Model* (TAM) didesain unuk menjelaskan faktor-faktor penentu penerimaan pelanggan dari berbagai teknologi komputasi *end user* (Davis, F.D., 1986). Model ini menduga bahwa penggunaan sistem yang sebenarnya ditentukan oleh niat perilaku pelanggan untuk menggunakan/*behavioral intention* (BI), yang dipengaruhi oleh sikap pelanggan/*attitude* (A) terhadap penggunaan. Akhirnya, A secara langsung dipengaruhi oleh keyakinan tentang sistem, yang terdiri PU dan PEOU. TAM berteori keyakinan-sikap-niat-hubungan perilaku untuk memprediksi penerimaan penggunaan teknologi.

Setelah pemeriksaan yang sesakma dari model dasr, ternyata mengungkapkan kelemahan yang diwarisi dari TAM. Sementara TAM telah sukses dalam memprediksi potensi penerimaan pelanggan, TAM memberikan sedikit bantuan pada desain dan pengembangan sistem dengan penerimaan tingkat tinggi. Toko virtual perlu mengatasi hambatan yang mencegah motivasi konsumen untuk membeli secara *online*. Untuk membuat model yang lebih kuat dan berarti bagi perancang sistem, kita perlu untuk mengidentifikasi faktor-faktor penentu PU dan PEOU.

Lei-Da Chen dan Justin Tan (Chen, Lei-Da, & Tan, Justin, 2004) mengambil pendekatan *critical success factor* (CSF) dan mengidentifikasi bidang utama agar toko virtual berkembang. Mereka menerapkan kombinasi teknik yang disarankan oleh Leidecker dan Bruno (Leidecker & Bruno, 1984) dan fous pada lima CSF untuk toko virtual penawaran produk, kekayaan infomasi, kegunaan dari etalase, kepercayaan yang dirasakan dan kualitas pelayanaan yang dirasakan.

**2.8.4.1 Penawaran Produk**

Persepsi konsumen tentang produk ternyata menjadi penentu utama dari belanja di retailer tertentu. Efikasi penawaran produk sering kali dinilai oleh tiga kriteria: luasnya pemilihan produk, strategi penetapan harga, dan kecocokan saluran produk retail. Kombinasi dari tiga faktor ini membentuk presepsi produk konsumen, yang diyakini untuk mempengaruhi PU dari toko virtual. Keanekaragaman produk seringkali merupakan faktor yang mempengaruhi dalam toko ritel (Woodside, A.G. & Trappey., 1992).

**2.8.4.2 Kekayaan Informasi**

Informasi yang memungkinkan pelanggan untuk mengklarifikasikan ambiguitas dan meningkatkan pemahaman terhadap masalah-masalah pada waktu yang tepat bisa dianggap informasi yang kaya. Kekayaan informasi memainkan peranan penting dalam membentuk keputusan konsumen untuk membeli dari toko virtual, namun karena masalah ketidakmampuan toko virtual untuk memenuhi kebutuhan pelanggan dalam menyentuh, mencium, atau mencoba produk sebelum dibeli mengurangi kualitas informasi yang diberikan oleh toko virtual. Oleh karena itu, toko virtual seringkali menggunakan taktik seperti perbandingan produk sebagai jalan untuk memperkaya informasi produk. Semakin banyaknya agen perbandingan produk *online* seperti *Excite's ProductFinder, Yahoo Shopping,* dan *MySimon.com* dapat merubah dinamika perdagangan di internet. Dalam rangka memprediksi kepuasan pelanggan terhadap produk secara akurat sebelum pembelian, konsumen online dapat mengharapkan informasi produk yang kaya dan fungsi perandingan produk yang kaya dan fungsi perbandingan produk yang kuat dari toko virtual (Saputra, Wulang R. A., 2014).

**2.8.4.3 Kegunaan Etalase**

Etalase sebuah toko virtual adalah *website* itu sendiri (Saputra, Wulang R. A., 2014). Desain etalase digital yang tidak bagus akan memberi pengaruh buruk pada pengalaman belanja *online* pelanggan, maka masalah *interface* yang berkaitan dengan navigasi, pencarian, dan proses pemesanan harus diberikan perhatian khusus, karena kemudahan pelanggan untuk mendapatkan apa yang dia inginkan diutamakan. Jika *website* mendapatkan nilai tinggi dalam kegunaannya, maka kemungkinan *website* tersebut duterima dan digunakan secara produktif sangat tinggi (Stahl, B., 1987). oleh karena itu kegunaan etalase menjadi faktor utama dalam penerimaan pelanggan terhadap toko virtual.

**2.8.4.4 Kualitas Pelayanan yang Dirasakan**

Lei-Da Chen dan Justin Tan (Chen, Lei-Da, & Tan, Justin, 2004) mengidentifikasikan lima dimensi yang digunakan konsumen untuk mengevaluasi kualitas layanan, yaitu *tangibles, reliability, responsiveness, assurance,* dan *empathy*. Kelima dimensi ini dijabarkan ke dalam konteks toko virtual sebagai beikut:

1. *Tangibles*, fasilitas fisik yang disediakan oleh toko virtual (penampilan toko virtual, keberadaan fasilitas layanan konsumen baik *online* maupun *offline*).
2. *Reliability*, kemampuan toko virtual untuk melakukan tindakan yang dijanjikan secara akurat dan bisa diandalkan (tepat waktu dan akurat dalam pengiriman produk)
3. *Responsiveness*, kesedian sebuah toko virtual untuk menawarkan bantuan kepada pelanggan di waktu yang tepat, misalnya dengan merespon secara ceoat *email* yang berisi pertanyaan konsumen
4. *Assurance*, kemampuan toko virtual untuk menginspirasi kepercayaan dan keyakinan
5. *Empathy,* meruapakan perhatian dan kepedulian kepada pelanggan yang diberikan oleh toko virtual (saran probadi pelanggan untuk produk).

**2.8.4.5 Kepercayaan yang Dirasakan**

Alasan mengapa banyak orang tidak belanja *online* adalah karena kurangnya kepercayaan terhadap bisnin *online*. Pada konteks belanja *online*, faktor yang mempengaruhi kurangnya kepercayaan pelanggan adalah pada informasi personal dan masalah keamanan data.

Suatu studi ((Hoffman, D.L., et al, 1996) dan (Greene, M.v., 1997)) menyatakan bahwa 42% responden meninggalkan *website* karena masalah privasi saat diminta untuk memberikan informasi pribadi. Faktor lain adalah pada masalah keamanan data, informasi yang dikirim melalui internet melintasi berbagai sistem komputer yang tidak aman,d an memunculkan risiko terjadinya intersepsi dan penyalahgunaan. Banyak pelanggan masih ragu untuk memberikan informasi pribadi, terutama dalam masalah finansial.

Jadi toko virtual harus dapat menunjukan komitmen untuk keamanaan data, karena kepercayaan konsumen hanya dapat terinspirasi jika risiko yang terkait dengan pembelian *online* dikurangi ke tingkat yang dapat ditoleransi oleh pelanggan (Saputra, Wulang R. A., 2014).

**2.9 Metode Pengembangan Sistem**

**2.9.1 Metode *SDLC***

*Software Development Life Cycle* (SDLC) menjelaskan metodologi untuk pengembangan berbagai macam *software*. Sebelum adanya SDLC, proses pengembangan *software* dilakukan melalui aktifitas yang tidak formal denagn aturan dan standar yang tidak formal. Hal ini mungkin menjadi awal mula dari berbagai masalah seperti penundaan dalam pengembangan, biaya operasional, dan kualitas *software* yang rendah. Pengenalan metode SDLC memberikan standar dan langkah dari pemngembangan *software*. Metode SDLC mengatasi segala masalah yang muncul sebelum metode ini ada (Saini, Munish & Kaur, Kuljit, 2014)

**2.9.1.1 Proses *Software***

Suatu proses merupakan kumpulan dari aktivitas, aksi, dan beberapa tugas yang dilakukan saat beberapa pekerjaan dalam sebuah produk dibuat. Sebuah aktivitas berusaha untuh meraih tujuan utama (misalnya komunikasi bersama beberapa orang yang berkepentingan) dan diterapkan terlepas dari domain aplikasi, ukuran proyek, kompleksitas usaha, atau tingkat ketelitian dengan yang teknik *software* terapkan. Suatu aksi (misalnya rancangan arsitektur) meliputi suatu himpunan pekerjaan yang dilakukan untuk membuat suatu produk (misalnya rancangan model arsitektur). Sebuah pekerjaan berfokus pada hal yang kecil, namun objek yang terdefinisikan dengan baik (misalnya memimpin sebuah unit pengujian) yang memproduksi keuntungan (Pressman, Roger S., 2002).

Kerangka kerja prosess mengimbangi dasar untuk melengkapi proses *software engineering* dengan mengidentifikasi jumlah kecil dari aktivitas kerangka kerja terapkan untuk seluruh projek *software*, tanpa memperhatikan ukuran mereka atau kompleksitasnya. Sebagi tambahan, proses kerrangka kerja meliputi suatu kumpulan dari aktivitas yang diterapkan melintasi keseluruhan proses *software*. Proses kerangka kerja secara umum bagi *software engineering* meliputi lima aktivitas (Pressman, Roger S., 2002) , antara lain :

1. *Communication*

Sebelum banyak teknis pekerjaan dimulai, sangat penting untuk mengkomunikasikan dan berkerja sama dengan klien (dan orang yang berkepentingan lainnya). Hal ini bermaksud untuk memahami tujuan dari pihak yang terkait dan untuk menghimpun kebutuhan yang membantu mendefinisikan fitur dan fungsi dari *software*.

1. *Planning*

Banyak perjalanan yang rumit akan menjadi lebih sederhana jika ada peta. Proyek *software* merupakan sebuah perjalanan yang rumit, dan perencanaan aktifitas menciptakan “peta” yang membantu membimbing tim pengembang dalam perjalanan tersebut. Peta-yang disebut rencana proyek *software*-mendefinisikan pekerjaan *software engineering* dengan menjelaskan tugas teknis untuk dilakukan, risiko yang mungkin terjadi, sumber daya yang tersedia, produk yang akan diproduksi, dan jadwal pekerjaan.

1. *Modeling*

Seorang penata taman, pembangun jembatan, insinyur penerbangan, tukang kayu, maupun arsitek, mereka bekerja menggunakan model setiap hari. Mereka mebuat “sketsa” berupa hal yang dapat dimengerti dalam gambaran besar, bagaimana komponen saling cocok satu sama lain, dan berbagai karakteristik lainnya. Jika bisa, anda dapat memperhalus sketsa menjadi lebih baik dan lebih detail dalam usaha untuk lebih memahami masalah dan bagaimana cara untuk menyelesaikannya. Seorang *software engineer* melakukan hal yang sama dengan menciptakan model untuk lebih memahami kebutuhan *software* dan rancangan yang melaksanakan kebutuhan tersebut.

1. *Contruction*

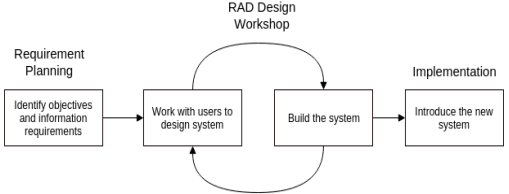
Aktivitas ini mengkombinasikan pembuatan kode (baik secara manual maupun otomatis) dan melakukan uji coba untuk mengatasi kesalahan kode.

1. *Deployment*

*Software* (sebagai entitas lengkap atau sebagai perkembangan yang lengkap sebagian) disajikan kepada klien yang mengevaluasi produk tersebut dan memberikan timbal balik berdasarkan evaluasi.

**2.9.1.2 *Rapid Application Development* (RAD)**

RAD merpakan proses yang mempercepat siklus pengembangan aplikasi. Fase ini serupa dengan metode *Waterfall* namun dengan 'potongan' yang lebih kecil. Penekanan pada model ini adalah pada iterasi cepat melaui suatu siklus. Prototip dirancang, dikembangkan dan dievaluasi oleh pengguna, melibatkanpengguna dalam proses dan mengoreksi desain tersebut. Model ini sangat cocok bagi proyek-proyek di perubahan lingkungan yang cepat, dimana tim perlu beradaptasi dengan berbagai situasi (Jenkins, Nick, 2008). Hal ini dimungkinkan untuk mengembangkan produk-produk yang berkualitas lebih cepat, sumber daya yang berharga ini dapat disimpan. Metode ini terdiri dari tiga tahap, yaitu : perencanaan kebutuhan, *workshop* perancangan RAD, dan implementasi (Noertjahya, Agustinus, 2002).

******

**Gambar 2.8 Fase RAD (Noertjahya, Agustinus, 2002)**

**2.9.2 Metode *Black Box***

*Black box testing* dapt disebut juga *functional testing*, meruapakan suatu teknik uji coba fungsional yang merancang kejadian berdasarkan informasi dari spesifikasi yang ada. *Blackbox testing* tidak memperhatikan mekanisme internal sistem, metode ini berfokus pada *output* yang dihasilkan dalam menganggapi masukan yang dipilih dan kondisi eksekusi. Dalam *black box testing, software teser* harus (atau tidak) memiliki akses ke *source code* internal itu sendiri (Nidra, Srinivas & Dondeti, Jagruthi, 2012). Kelebihan penggunaan *black box testing* antara lain (Khan, M. Ehmer & Khan, Farmeena, 2012) :

1. Efisien untuk *code segment* yang luas
2. Tanggapan penguji yang sangat sederhana
3. Sudut pandang pengguna terpisah dari sudut pandang pengembang (*programmer* dan *tester* tidak berikatan satu sama lain).
4. Pengembangan penujian kasus yang lebih cepat.

**2.10 Metode Pengumpulan Data**

**2.10.1 Observasi**

Observasi dilkakukan dengan melakukan pengamatan terhadap enam *website e-commerce* penjualan kosmetik, yaitu The Balm Indonesia, The Body Shop Indonesia, L'Occitane, Sephora Indonesia, MIMO Beauty dan Sociolla dengan meneliti tiga fitur pencarian produk, yaitu *autocomplete, autocorrect* dan *autosuggestion.*

**2.10.2 Studi Pustaka**

Pengumpulan data yang bersumber dari berbagai penelitian seperti dalam artikel, jurnal, buku dan sumber lain dari internet yang menunjang pemecahan masalah.

**2.11 Alat Bantu Pemodelan Sistem**

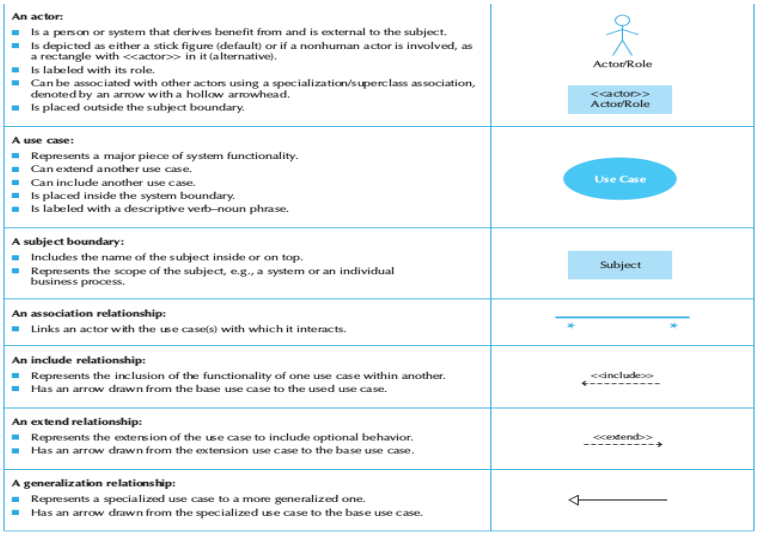
**2.11.1 *Unified Modelling Language* (UML)**

*Unified Modeling Language* (UML) adalah bahasa pemodelan visual dengan tujuan yang umun yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasi artefak pada sistem *software*. UML membuat keputusan dan memahami sistem yang harus dibangun. UML digunakan untuk memahami, merancang, menelusuri, mengkonfigurasi, memelihara, dan mengontrol informasi sistem. Bahasa pemodelan ini dimaksudkan untuk menyatukan pengalaman masa lalu tentang teknik pemodelan dan untuk menggabungkan praktek *software* terbaik saat ini ke dalam pendekatan standar (Rumbaugh et al, 2005).

UML memperoleh informasi mengenai struktur statis dan tingkah laku dinamis dari sistem. Sistem dimodelkan sebagai koleksi objek-objek diskrit yang berinteraksi untuk melakukan pekerjaan yang pada akhirnya menguntungkan pengguna di luar. Struktur statis mendefinisikan jenis objek penting untuk sistem dan untuk implementasinya, serta hubungan antara objek-objek. Perilaku dinamis mendefinisikan mendefinisikan sejarah objel-objek dari waktu ke waktu dan komunikasi antara objek-objek untuk mencapai tujuan. Pemodelan sistem dari beberapa sudut pandang yang terpisah namun terkait mengizinkan untuk memahami tujuan yang berbeda (Rumbaugh et al, 2005).

**2.11.1.1 *Use Case Diagram***

*Use Case Diagram* merupakan sdalah satu dari diagram UML yang digunakan untuk menggambarkan fungsionalitas sistem. Diagram ini menampilkan fungsionalitas yang sistem sediakan dan pengguna yang akan berkomunikasi dengan sistem untuk menggunakan fungsionalitas yang ada. Diagram ini terdiri dari dari *actor, use cases,* komunikasi *assosiation* dan batasan sistem atau subsistem. Diagram ini menyajikan interaksi antara *actor* dan sistem untuk meraih tujuan tertentu. Manusia atau sistem eksternal keduanya diasumsikan sebagi a*ctor* dalam diagram ini. *Use cases* mendefinisikan fungsionalitas sistem dari prespektif pengguna dan juga digunakan untuk mendokumentasikan ruang lingkup sistem. Biasanya, *use cases* digunakan pada teknik sistem level yang lebih tinggi sebagai perbandingan untuk *use cases* dalam *software engineering*. Sintaks dari diagram ini dapat dilihat dalam Gambar 2.7.



**Gambar 2.9 Sintaks *Use Case Diagram* (Dennis et al, 2005)**

*Actor* merupakan idealisasi dari peran serta atas pihak eksternal, proses, atau sesuatu yang berinteraksi bersama sistem, sumsistem, atau *class*. *Actor* menggambarkan interaksi bahwa *class* daro pengguna luar mungkin berinteraksi dengan sistem. Pengguna yang berbeda dapat terikat dengan *actor* yang sama dan karena itu mewakili beberapa contoh definisi *actor* yang sama (Rumbaugh et al, 2005).

Setiap *actor* berpartisipasi terhadap satu atau lebih *case*. *Actor* berinterkasi dengan *use case* dengan saling bertukar pesan. Implementasi internal *actor* tidak relevan dalam *use case;* *actor* dapat dicirikan cukup dengan himpunan atribut yang mendefinisikan keadaannya. Seorang *actor* digambarkan semagai *stick man* kecil dengan nama dibawahnya.

**2.11.1.2 *Activity Diagram***

Sebuah aktivitas merupakan urutan operasi dari permulaan menuju akhir dari sistem dan tiap aktivitas dapat ditransfomasikan pada proses data (Azizi, Somayeh, 2011). *activity diagram* merupakan representasi grafis dari alur kerja pada setiap langkah aktivitas dan aksi bersama dengan dukungan pilihan, perulangan dan konkurensi. Dalam UML, *activity diagram* dapat digunakan untuk menjelaskan bisnis dan alur kerja langkah demi langkah operasional dari komponen dalam suatu sistem. *Activity diagram* menampilkan keseluruhan alur dari kontrol (Contantianus, F. & Suteja, B.R., 2005).

**2.11.2 Struktur Navigasi**

Struktur navigasi adalah plot yang digunakan dalam aplikasi yang akan dibuat. Sebelum mempersiapkan aplikasi multimedia dalam *software*, menentukan aliran yang akan digunakan dalam aplikasi yang dibuat garus dilakukan pertama kali. Bentuk dasar dari struktur navigasi pada umumnya digunakan dalam pembuatan proses yang terdapat empat jenis apliaksi multimedia, yaitu struktur navigasi liniear, non-linear, hirarkim dan campuran (Mahendra, A., 2016).

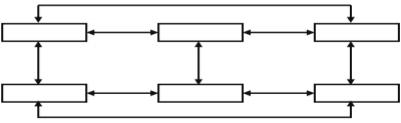
**2.11.2.1 Struktur Navigasi Linier**

Struktur navigasi linier adalah struktur yang memiliki rangkaian cerita yang berurutan. Struktur ini menampilkan lapisan satu demi satu secara berurutan, sesuai aturan (Mahendra, A., 2016).

**Gambar 2.10 Struktur Navigasi Linier (Mahendra, A., 2016)**

**2.11.2.2 Struktur Navigasi Non Linier**

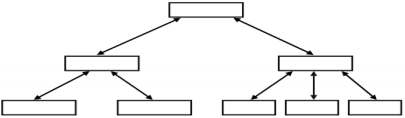
Struktur navigasi non liner (tidak berurutan) merupakan pengembangan dari struktur navigasi linier, hanya saja pada struktur navigasi ini dibolehkan mebuat cabang. Percabangan pada struktur ini berbeda dari percabangan pada struktur navigasi hirarki. Pada struktur navigasi non linier, posisi dari seluruh halaman sama, sehingga tidak terdapat halaman utamanya (Mahendra, A., 2016).



**Gambar 2.11 Struktur Navigasi Non Linier (Mahendra, A., 2016).**

**2.11.2.3 Struktur Navigasi Hirarki**

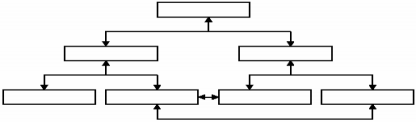
Struktur navigasi hirarki yang biasa disebut juga struktur navigasi bercabang, merupakan struktur bercabang yang bergantung pada data atai gambar untuk ditampilkan pada suatu lapisan dengan kriteria tertentu. Tampilan pada menu utama disebut halaman utama, yang memiliki halaman yang disebut halaman cabang dan jika dipilih akan berada pada halaman kedua, dan seterusnya (Mahendra, A., 2016).



**Gambar 2.12 Struktur Navigasi Hirarki (Mahendra, A., 2016).**

**2.11.2.4 Struktur Navigasi Campuran**

Struktur navigasi campuran adalah kombinasi dari struktur navigasi sebelumnya dan juga disebut struktur navigasi bebas; intinya adalah jika layarmembutuhkan percabangan kemudian akan dibuat percabangan. Struktur ini paling banyak digunkaan dalam pembuatan aplikasi multimedia.



**Gambar 2.13 Struktur Navigasi Hirarki (Mahendra, A., 2016).**

**2.12 *Django***

Untuk membuat aplikasi *website* ini diperlukan *website* *framework* *Django*. Pengertian dari *framework* sendiri menurut Syakir Rahman dalam *website-*nya, menyebutkan *framework* adalah sebuah software untuk memudahkan para programmer membuat aplikasi atau *website* yang berisi berbagai fungsi, *plug-in*, dan konsep sehingga membentuk suatu sistem tertentu. Dengan menggunakan framework, sebuah aplikasi akan tersusun rapi dan terstuktur.

Menurut Dewi et al. (2012:21) dalam jurnalnya mengungkapkan bahwa *Django* adalah sebuah *website framework* yang berbasis Python dan menggunakan sistem MTV, yaitu *Model, Template* dan *View*.*Website framework* itu sendiri adalah sebuah alat yang digunakan untuk mempermudah kita dalam membangun sebuah website. *Model* adalah layer yang digunakan untuk berinteraksi dengan database, *Template* adalah layer presentasi untuk HTML, XML dan lainnya, sedangkan *View* adalah *layer* yang berisikan logika yang mengolah data dari model dan mengirimkannya ke dalam Template.

Selain itu bahasa pemrograman Python sendiri menurut Hudaya (2013: 1) Python adalah salah satu bahasa pemrograman *open source* yang ringkas, sederhana, dan bisa digunakan di beberapa sistem operasi. Python bersifat gratis dan bisa diunduh oleh siapa saja melalui www.python.org dengan berbagai versi yang ada.

Django merupakan kerangka kerja aplikasi berbasis *website* Python level tinggi yang dirancang untuk membantu pengembangan *website* dinamis, aplikasi berbasis *website*, dan *web services*. Django dirancang untuk mengenalkan pengembangan secara cepat dan bersih, rancangan pragmatis dan membangun aplikasi berbasis *website* yang *high-performing* dan elegan secara cepat. Keuntungan utama dari Django (Hourieh, Ayman, 2008) antara lain :

1. Integrasi ketat antar komponen

Pertama-tama, Django menyediakan satu set komponen terintegrasi; seluruh komponen ini telah dikembangkan oleh tim Django itu sendiri. Django pada walnya dikembangkan sebagai kerangka kerja *in-house* untuk mengelola serangkaian situs-situs berbasis berita. Kemudian kode dirilis di internet dan tim Django meneruskan perkembangannya menggunakan model *open source*. Karena akrnya, komponen Django dirancang untuk integrasi, usabilitas dan kecepatan di permulaan.

1. *Object-Relational Mapper*

Komponen *database* Django, *Object Relational Mapper* (ORM), menyediakan penghubung antara model data dan mesin *database*. ORM mendukung suatu himpunan besar sistem *database*, dan beralih dari satu mesin ke mesin lain dilakukan untuk mengubah *file* konfigurasi. Hal ini meberikan fleksibilitas yang besar jika keputusan dibuat untuk mengubah dari satu mesin *database* ke mesin lainnya.

1. Rancangan URL yang bersih

Sistem URL pada Django sangat fleksibel dan efektif, sistem ini mempersilakan pengembang aplikasi untuk mendefinisikan pola URL, dan mendefinisikan fungsiPython untuk menangani tiap pola yang ada. Sistem ini juga membuat pengembang aplikasi untuk membuat URL yang ramah terhadap pengguna dan mesin pencari.

1. Antarmuka administrasi yang otomatis

Django hadir bersama denagn antar muka administrasi yang siap untuk dipakai. Antarmuka ini membuat pengelolaan data aplikasi semakin mudah. Antarmuka halaman administrasi ini juga sangat fleksibel dan mudah diubah sesuai keinginan.

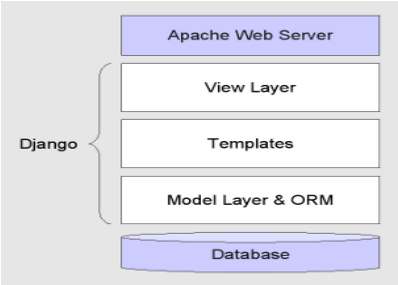
1. Lingkungan pengembangan yang lebih baik

Sebagai tambahan, Django menyediakan lingkungan yang sangat baik untuk pengembangan aplikasi. Lingkungan ini hadir dengan *server* *website* yang ringan untuk pengembangan juga uji coba aplikasi. Ketika mode *debuging* diaktifkan, Django menyediakan pesan kesalahan yang sangat teliti dan mendetail dengan berbagai informasi *debugging*. Semua fitur ini mepermudah perbaikan kesalahan pada kode.

1. Bantuan multi bahasa

Django mendukung *website* multi bahasa melalui *buit-in* sistem internasionalisasi. Hal ini bisa sangat berharga bagi mereka yang berkerja di *website* dengan lebih dari satu bahasa. Sistem ini menerjemahkan antarmuka menjadi tugas yang sederhana.

Konsep Django agak mirip dengan rancangan *Model View Controller,* tetapi Django menjelaskannya sebagai kerangka kerja *Model Template View* (MTV). Model Django dimodifikasi sedikit untuk mewakili struktur aplikasi *website*. Lapisan *View* terdiri dari *business logic* dan *request handling*. *Template* menentukan tampilan *website*, dan lapisan Model dan ORM mendefinisikan data dan relasi yang disimpan dalam *database* (Ward, Ian, 2009).



**Gambar 2.14 Lapisan Pada Django (Ward, Ian, 2009)**

**2.13 Elasticsearch**

Elasticsearch adalah mesin pencari *real-time* terdistribusi dan mein penganalisa. Elasticsearch memungkinkan penggunanya mengeksplorasi data dengan kecapatan dan skala yang belum pernah ada sebelumnya. Elasticsearch digunakan untuk pencarian *full-text*, pencarian terstruktur, analisis, dan konbinasi dari ketiganya seperti berikut ini :

* Wikipedia menggunakan Elasticsearch untuk menyediakan pencarian *full-text* dengan *highlighted search snippets*, dan pencarian berdasarkan kata kunci yang diberikan serta pemberian saran yang dimaksud pengguna*.*
* The Guardian menggunakan Elasticsearch untuk menggabungkan *log* pengunjung dengan data *social-network*, untuk memberikan umpan balik secara *real-time* ke editor tentang respon publik terhadap artikel baru.
* Stack Overflow mengkombinasikan pencarian *full-search* dengan *geolocation query* dan pengguna yang serupa dengan pengguna yang dimaksud untuk menemukan pertanyaan dan jawaban yang sejenis
* GitHub menggunakan Elasticsearch untuk melakukan *query* sebanyak 130 miliar baris kode .

Namun, Elasticsearch tidak hanya dipakai oleh korporasi besar saja. Elasticsearch juga memungkinkan banyak perusahaan baru seperti Datadog dan Klout untuk membuat prototipe ide dan untuk mengubahnya menjadi solusi yang *scalable*. Elasticsearch juga dapat dijalankan pada *Personal Computer* (PC) seperti *laptop* maupun dalam skala besar seperti ratusan server dengan ukuran data *petabyte*.

Tidak ada bagian dari Elasticsearch yang baru atau revolusioner, pencarian *full-text* telah dilakukan sebelumnya, seperti memiliki sistem analisis dan *database* terdistribusi. Revolusi yang dilakukan Elasticsearh yaitu mengkombinasikan bagian individu tersebut menjadi aplikasi satu kesatuan yang koheren dan *real-time.*

Elasticsearch merupakan mesin pencari *open-source* yang dibangun pada Apache Lucene, yang merupakan *library* mesin pencari secara *full-text*. Lucene bisa dibilang *library* yang paling canggih dengan kinerja yang tinggi, dan *library* mesin pencari dengan fitur lengkap.

Namun Lucene hanyalah sebuah *library*. Untuk meningkatkan kekuatannya, pengguna perlu untuk melakukan pekerjaannya di Java dan untuk mengintregasikan Lucenesecara langsung dengan aplikasi yang telah dibuat. Lebih uruk lagi karena pengguna akan memerlukan *degree* dalam pencarian informasi untuk memahami cara kerjanya, karena Lucene sangat kompleks.

Namun, Wlasticsearch jauh lebih dari sekedar Lucene dan lebih dari “hanya” pencarian *full-text*. Hal ini dapat digambarkan sebagai berikut :

* Penyimpanan dokumen terdistribusi secara *real-time* dimana setiap *field* diindeks dan dapat dicari.
* Mesin pencari terdistribusi dengan analisis *real-time*.
* Mampu untuk *scaling* ratusan *server* dan data baik terstruktur maupun tidak terstruktur dengan ukuran mencapai *petabyte.*

Keseluruhan fungsionalitas tersebut dikemas ke *standalone server* dan memberitahu bahwa aplikasi yang telah dibuat pengguna dapat berbicara dengan melalui RESTful API sederhana, enggunkan suatu *web client* dari bahasa pemrograman yang telah dipakai pengguna, atau bahkan dapat dilakukan melalui *command line*.

**2.13.1 Terhubung dengan Elasticserach**

Selain menggunakan Java API, pengguna juga dapat menggunakan Restful API bersama dengan JSON untuk dapat terhubung dengan Elasticsearch melalui HTTP. Semua bahasa pemrograman lain juga dapat berkomunikasi dengan Elasticsearch melalui *port* 9200 menggunakan RESTful API, yang dapat diakses dengan *web client*. Bahkan pengguna dapat terhubung dengan Elasticsearch melalui *command line* menggunakan *curl command*.

**2.13.2 Pencarian pada Elasticsearch**

**2.13.2.1 Pencarian Terstruktur**

**2.13.2.2 Pencarian *Full-Text***

**2.13.2.3 Pencarian *Multifield***

**2.13.2.4 Proximity Matching**

**2.13.2.5 Partial Matching**

**2.13.2.6 Mengkontrol Relevansi**

**2.13.3 Pemrosesan Bahasa Alami pada Elasticserach**

**2.13.3.1 Mengidentifikasi Kata**

**2.13.3.2**

**2.14 SQLite**

SQLite merupakan sebuah sistem manajemen basis data relasional yang bersifat ACID - compliant dan memiliki ukuran pustaka kode yang relatif kecil, ditulis dalam bahasa C (Nugroho, Dimas R.S., 2011).

SQLite merupakan proyek yang bersifat *public domain* yang dikerjakan oleh D. Richard Hipp. Tidak seperti pada paradigma *client-server* umumnya, inti SQLite bukanlah sebuah sistem yang mandiri yang berkomunikasi dengan sebuah program, melainkan sebagai bagian integral dari sebuah program secara keseluruhan. Sehingga protokol komunikasi utama yang digunakan adalah melalui pemanggilan API secara langsung melalui bahasa pemrograman. Mekanisme seperti ini tentunya membawa keuntungan karena dapat mereduksi *overhead*, latency times, dan secara keseluruhan lebih sederhana. Seluruh elemen basisdata (definisi data, tabel, indeks, dan data) disimpan sebagai sebuah file. Kesederhanaan dari sisi disain tersebut bisa diraih dengan cara mengunci keseluruhan file basis data pada saat sebuah transaksi dimulai (Nugroho, Dimas R.S., 2011).

SQLite mengimplementasikan hampir seluruh elemenelemen standar yang berlaku pada SQL-92, termasuk transaksi yang bersifat *atomic*, konsistensi basisdata, isolasi, dan durabilitas (dalam bahasa inggris lebih sering disebut ACID), *trigger*, dan kueri-kueri yang kompleks. Tidak ada pengecekan tipe sehingga data bisa dientrikan dalam bentuk string untuk sebuah kolom bertipe integer. Beberapa kalangan melihat hal ini sebagai sebuah inovasi yang menambah nilai guna dari sebuah basisdata, utamanya ketika digunakan dalam bahasa pemrograman berbasis script (PHP, Perl), sementara kalangan lain melihat hal tersebut sebagai sebuah kekurangan. Beberapa proses ataupun thread dapat berjalan secara bersamaan dan mengakses basis data yang sama tanpa mengalami masalah. Hal ini disebabkan karena akes baca data dilakukan secara paralel. Sementara itu akses tulis data hanya bisa dilakukan jika tidak ada proses tulis lain yang sedang dilakukan; jika tidak, proses tulis tersebut akan gagal dan mengembalikan kode kesalahan (atau bisa juga secara otomatis akan mencobanya kembali sampai sejumlah nilai waktu yang ditentukan habis). Hanya saja ketika sebuah tabel temporer dibuat, mekanisme penguncian pada proses multithread akan menyebabkan masalah . *Update* yang terkini (versi 3.3.4) dikatakan telah memperbaiki masalah ini. Sebuah program yang mandiri dinamakan sqlite disediakan dan bisa digunakan untuk mengeksekusi kueri dan memanajemen file-file basisdata SQLite. Program tersebut juga merupakan contoh implementasi penulisan aplikasi yang menggunakan pustaka SQLite (Nugroho, Dimas R.S., 2011).

**2.15 RESTful API**

*Representational State Transfer* (REST) diperkenalkan sebagai gaya arsitektur untuk membangun system hypermedia berskala besar. Gaya arsitektur ini. Gaya arstektur ini merupakan entitas yang agakabstrak, dimana prinsip-prinsip yang telah digunakan untuk menjelaskan skalabilitas yang sangat baik dari protokol HTTP 1.0 dan juga telah dibatasi desain versi berikutnya, yaitu HTTP 1.1. Jadi, istilah REST sangat sering dihubungkan dengan HTTP (Pautasso et al., 2008).

Gaya arsitektur RESt berdasarkan empat prinsip berikut ini :

1. Identifikasi Sumber Daya Melalui URI.

RESTful *web service* memperlihatkan seperangkat sumber daya yang mengidentifikasi target dari interaksi dengan *client*. Sumber daya diidentifikasi oleh URI (Berners-Lee et al., 2005) yang memberikan ruang pengalamatan global untuk penemuan sumber daya dan layanan (Pautasso et al., 2008).

1. Antarmuka yang Seragam

Sumber daya yang dimanipulasi menggunakan himpunan tetapdari empat operasi, yaitu *create, read, update,* dan *delete* menjadi operasi: PUT, GET, POST dan DELETE. Operasi PUT menciptakan sumber dyaa baru yang dapat dihapus kemudian menggunakan operasi DELETE. Operasi GET mengambil keadaan saat sumber daya pada beberapa representasi. Operasi POST mentransfer *state* baru ke sumber daya (Pautasso et al., 2008).

1. *Self-Descriptive Messages*.

Sumber daya dipisahkan dari representasi mereka sehingga konten dari sumber daya tersebut dapat diakses dalam berbagai format (misalnya HTML, XML, *plain text*, PDF, JPEG, dan lain sebagainya). Metadata pada sumber daya yang tersedia digunakan, misalnya untuk mengontrol *caching*, mendeteksi kesalahan transmisi, menegosiasikan format representasi yang tepat, dan melakukan otentikasi atau akses kontrol (Pautasso et al., 2008).

1. Interaksi *Stateful* Melalui *Hyperlink*.

Setiap interaksi dengan sumber daya itu *stateless*, yaitu pesan permintaan yang mandiri. Interaksi *stateful* didasarkan pada konsep transfer *state* eksplisit. Beberapa teknik yang ada untuk bertukar *state*, misalnya menulisan ulang URI, *cookies*, dan *form* tersembunyi. *State* dapat tertanam dalam menanggapi pesan untuk menunjuk ke *state* yang selanjutnyadari interaksi (Pautasso et al., 2008).

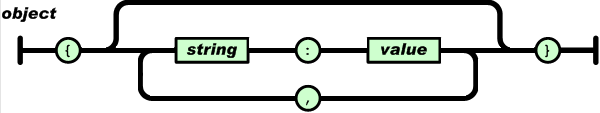
**2.16 JSON**

JSON(*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari [Bahasa Pemprograman JavaScript](http://javascript.crockford.com/), [Standar ECMA-262 Edisi ke-3 - Desember 1999](http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf). JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dan lain-lain. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data (International, Ecma, 2013). JSON terbuat dari dua struktur:

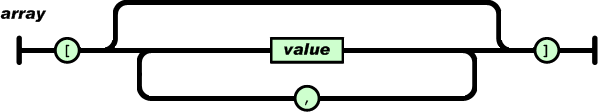
* Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau [*associative array*](http://en.wikipedia.org/wiki/Associative_array).
* Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut (International, Ecma, 2013):

1. Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan **{** (kurung kurawal buka) dan diakhiri dengan **}** (kurung kurawal tutup). Setiap nama diikuti dengan **:** (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh **,** (koma).

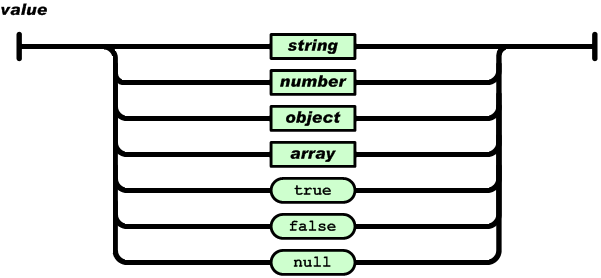
**Gambar 2.15 Objek pada JSON (International, Ecma, 2013)**

1. Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [ (kurung kotak buka) dan diakhiri dengan ] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).

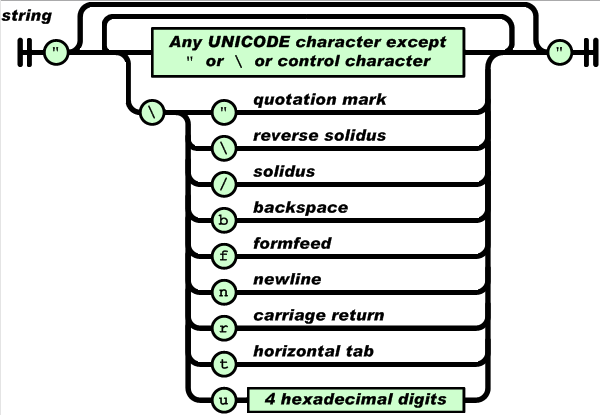


**Gambar 2.16 Larik pada JSON (International, Ecma, 2013)**

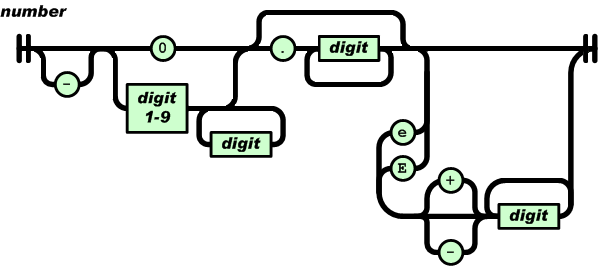
1. Nilai (*value*) dapat berupa sebuah string dalam tanda kutip ganda, atau *angka*, atau *true* atau *false* atau *null*, atau sebuah *objek* atau sebuah *larik*. Struktur-struktur tersebut dapat disusun bertingkat.

**Gambar 2.17 *Value* pada JSON (International, Ecma, 2013)**

1. String adalah kumpulan dari nol atau lebih karakter *Unicode*, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.

**Gambar 2.18 String pada JSON (International, Ecma, 2013)**

1. Angka pada JSON sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



**Gambar 2.19 Angka pada JSON (International, Ecma, 2013)**

Spasi kosong (*whitespace*) dapat disisipkan di antara pasangan tanda-tanda tersebut, kecuali beberapa detil *encoding* yang secara lengkap dipaparkan oleh bahasa pemprograman yang bersangkutan (International, Ecma, 2013).

**2.17 HTML**

HTML merupakan *file* teks murni yang dapat dibuat dengan berbagai macam *text editor*. HTML adalah suatu bahasa untuk mendeskripsikan struktur halaman web. HTML memberikan bermakna untuk (Rabbani, M.H., 2015) :

* Mempublikasikan dokumen secara *online* dengan  *headings,* teks, tabel, *list*, foto, dan lain sebagainya.
* Mengambil informasi secara online melalui l*ink hypertext* dan pada tombol.
* Merancang *form* untuk melakukan transaksi dengan layanan jarak jauh, untuk digunakan dalam mencari informasi, membuat pemesanan, pemesanan produk, dan lain sebagainya.
* Termasuk spreadsheet, klip video, klip suara dan sistem lain secara langsung dalam dokumen.

Namun demikian, sampai saat ini HTML masih berdiri kokoh sebagai dasar bahasa web seperti PHP, ASP, JSP dan lainnya. Bahkan, secara umum, sebagian besar situs di Internet masih menggunakan HTML sebagai teknologi utama (Constantianus, Frederick & Suteja, B.R., 2005).

**2.18 Studi Sejenis**

Untuk perbandingan dan sebagai acuan dalam implementasi pengembangan mesin pencari, maka dilakukan pengamatan terhadap penelitian sejenis yang telah dilakukan sebelumnya. Dari hasil pengamatan didapatkan beberapa literatur penelitian sejenis yang dapat diklasifikasikan sebagai berikut.

Penerapan Semantic Similarity Pada Fasilitas Pencarian *Website E-Commerce* , Wulang Risky Aprianto Saputra, 2014, Universitas Gunadarma : Jakarta. Pada skripsi ini penulis membuat situs *e-commerce* yang sesuai dengan aspek Interaksi Manusia dan Komputer (IMK) menggunkan *Adobe Dreamweaver CS5* sedangkan *database* untuk *semantic similarity* menggunakan MySQL. Situs ini berfokus pada penjualan produk-produk IT. Skripsi menggunakan *query* rewriting yang merupakan *semantic* similarity, sehingga produk yang ditemukan pelanggan saat proses pencarian akan dialihkan ke produk lain yang sejenis. Kekurangan mesin pencari pada skripsi ini yaitu pengguna hanya dapat memasukan satu kata kunci pada mesin pencari.

Penelitian saat ini pada mesin pencari berfokus pada metodologi yang diadopsi untuk  *Semantic Search Engines* (SSEs)berdasarkan ontologi dengan penekanan pada domain tertentu dan peningkatan kinerjanya. Dalam konteks ini, terdapat beberapa hasil survei yang luas pada SSEs (Piraviperumal et al, 2013). Ilyas et al. (llyas et al., 2004) mengusulkan arsitektur konseptual untuk SSE dengan fokus utama pada mesin inferensi. Inferensi dapat didefinisikan sebagai proses yang berasal dari kesimpulan logis berdasarkan laporan tertentu yang diketahui atau telah dianggap benar. Pada *web* semantik, menemukan hubungan baru atau pengetahuan dari data *website* dapat disebut sebagai inferensi (W3C, 2012). Model yang diusulkan Ilyas et Al. Menyediakan basis pengetahuan yang lengkap yang dibuat menggunakan *database* relasional. Karena basis pengetahuan ini, mereka mengkaim bahwa metode mereka menunjukan kinerja yang tinggi karena data dapat secara langsung melakukan *query* dari sana, daripada data konvensional. Namun, arsitektur ini mengasumsikan bahwa penjelasan data *website* seudah di tangan yang tidak benar, maka hal ini merupakan keterbatasan utama dari metode ini.

Zi-Qiang et al. (Zi-Qiang et al., 2007) telah mengusulkan kerangka mesin pencari web semantik berbasis ontologi. Arsitektur terdiri atas mesin pencari semantik dimana web di-*crawl.* Selanjutnya, mesin *ratiocination* digunakan untuk menganalisa *query* dari pengguna menggunakan *semantic reasoning* berbasis padaontologi. Zi-Qiang et al. Membuat model prototipe mesin pencari semantik menggunakan Lucene, dan hasilnya menunjukkan validitas yang efektif dan daya ingat validitas semantik yang baik, presisi dan tingkat pemulihan diatas 90% untuk permintaan semantik dalam ontologi yang diberikan. Di sisi lain, ia mengamati bahwa hasil pencarian tidak terlalu baik jika semantik dari kebutuhan pengguna di luar jangkauan ontologi yang dibangun. Selanjutnya, proses mengukur kesamaan ontologi dalam proses *query* juga tidak didefinisikan dengan baik (Piraviperumal et al, 2013).

Mukhopadhyay et al. (Mukhopadhyay et al., 2007) menyajikan suatu spesific domain ontologi berdasarkan mesin pencari semantik yang dikembangkan untuk *website* pertanian yang menjadi *host* bagi informasi pertanian tentang wilayah Bengal barat. Hasil kerja mereka menghasilkan rancangan metodologi dan pengembangan mesin pencari web semantik yang terutama berfokus pada hasi pencarian dengan presisi tinggi untuk domain pencarian tertentu. Metode mereka menghilangkan hasil yang tidak relevan dengan mengguanakan teknik pemetaan antara contoh dan kelas. Arsitektur ini sangat berguna ketika salah satu memiliki repositori tetap dari halaman web untuk mendapatkan informasi darinya. Namun, ketika pencarian didistribusikan dengan format data yang berbeda dalam repositori yang berbeda dan *database* yang berbeda, maka arsitektur ini mungkin tidak menguntungkan (Piraviperumal et al, 2013).

Dong et al. (Dong et al., 2008) mengusulkan suatu ontologi layanan transportasi untuk membantu menyelesaikan masalah layanan transportasi yang tidak efisien. Mereka merancang algoritma pencarian semantik berdasarkan algoritma *Case-Based Reasoning* (CBR) untuk implementasi SSE berbasis pengetahuan menggunakan konsep *threshold* untuk menentukan peringkat *output* menjadi hasil yang konkret. Keterbatasan metode mereka adalah bahwa sumber data dari metadata layanan transportasi telah terbatas untuk *Australian Yellow Pages*, dimana menyediakan ketersedian sumber daya yang terbatas. Selain itu, karena kekurangan penjelasan data, konsep ni tidak dapat disesuaikan untuk web (Piraviperumal et al, 2013).

Maziz Esa et al. (Maziz Esa et al., 2010) mengembangkan model ekstensi mesin pencari bernama Zenith, yang merupakan *pluh-in* untuk Nutch, yang memungkinkan untuk berfungsi sebagai mesin pencari semantik. Arsitektur ini sangat modular dan komponen dari proyek sebelumnya dapat digunakan kembali dalam pembangunan Zenith. Kekurangan arsitektur campuran iniadalah bahwa hal itu mengakibatkan terganggunya sistem penilaian Nutch, dan karena itu lah tidak dapat menampilkan hasil sesuai dengan peringkat yang penting. Namun, analisis dan pembahasan dalam makalah tersebut tetap non-konklusif (Piraviperumal et al, 2013).

Qu et al. (Qu et al., 2011) mengusulkan arsitektur dan desain sistem pencarian berdasarkan wen semantik. Sistem mereka didasarkan pada Jema, sebuah kerangka kerja Java untuk membangun aplikasi semantik web. Namun, metode mereka memiliki beberapa kekurangan yang melekat seperti pembuatan ontologi manual, hanya menggunakn mesin inferensi milik Jena dan lain sebagainya karena dipengaruhi kinerja sistem sekaligus mengoptimalkan pemulihan dan tingkat presisi. Jin et al. (Jin et al., 2008) mengusulkan algoritma berdasarkan pada algoritma *Term Frequency/Inverse Document Frequency* (TF/IDF) untuk menjamin pengambilan sumber daya informasi dengan cara yang lebih efisien. Pengembangan algoritma TF/IDF tergantung pada *quoted frequency* dari kata kunci di halaman web dan tag semantik dalam isi dari halaman web. Pebaikan algoritma ini mempromosikan efisiensi dan ketepatan mesin pencari web tradisional dan mesin pencari web semantik. Sinha et al. (Sinha et al., 2012) mengembangkan model domain spesifik yang mendukung beberapa domain sehingga memberikan skalabilitas tinggi. Prototipe ini terutama dirancang untuk beberapa produk yang umum digunakan misalnya buku, ponsel, obat-obatan, perhiasan dan lain sebagainya. Model pencarian menyimpan konten RDF untuk tiap domain spesifik halaman web yang ada di domain repositori halaman web tertentu dan menawarkan *output* yang lebih lengkap dan komprehensif (Piraviperumal et al, 2013).

Nasraoui dan Zhuhadar (Nasraoui & Zhuhadar, 2010) menyajikan mesin pencari dengan *recall* dan *recision* tingkat tinggi dan mengimplementasikan arsitekturini pada platform yang disebut *HyperManyMedia*. Protégé, editor OWL yang *open source*, digunakan untuk merancang dan membangun struktur ontologi *HyperManyMedia*. Terdapat dua jenis yang berbeda dari ontologi yang digunakan dalam sistem ini, model ontologi global yang mewakili seluruh domain *E-learning*, dan model pelajar yang mewakili profil pelajar. Selanjutnya mesin pencari memberikan peringkat ulang tidak hanya berdasarkan profil pengguna, namun juga pada *cluster-based similarity metrics* yang menangkap distribusi dokumen dalam domain (Piraviperumal et al, 2013).

Shekarpour et al. (Shekarpour et al., 2009) mengusulkan teknik untuk secara otomatis mengubah *query* kata kunci ke SPARQL karena kebanyakan web aplikasi data fokus utamanya menggunakan SPARQL untuk mengeluarkan *query*. Shekarpour et al. juga bekerja pada peningkatan metode ini untuk menerapkannya pada subset besar dari *Linked Data Web*. Namun, mencapai kinerja yang cukup untuk menjawab *query* secara instan menjadi masalah yang lebih besar dalam kasus RDF, diamna pendekatan mereka saat ini terbatas pada dua kata kunci. Masalh-masalah ini dapat memburuk jika algoritma ini diterapkan pada skala besar dan pada basis pengetahuan yang terdesentralisasi. Menggunakan pendekatan ini juga dapat mengakibatkan kesulitan dalam mengakses data web untuk pengguna yang non-ahli (Piraviperumal et al, 2013).

Beberapa proyek menarik saat ini dalam pengembangan yang aktif. Dbpedia dan Swoogle merupakan dua diantaramya. DBpedia (Dbpedia, 2012) merupakan proyek yang bertujuan untuk mengekstrak koten terstruktur dari informasi yang dibuat sebagai bagian dari proyek Wikipedia. Informasi terstruktur ini kemudian tersedia untuk aplikasi lain pada *World Wide Web.* Swoogle (Swoogle, 2004) merupakan mesin pencari metadata untuk web semantik, ontologi, dokumen, *terms,* dan data yang dipublikasikan dalam web. Swoogle menggunakan satu set *crawler* untuk menemukan dokumen RDF dan juga dokumen HTML dengan tambahan konten RDF. Hal ini beralasandokumen-dokumen ini bersama dengan bagia penyusunnya, mencatat dokumen-dokumen dan indeks metadata bermakna tentang dokumen dalam *database*-nya. Pengguna manusia mengakses layanan Swoogle melui akses antarmuka *browser* dan perangkat lunak akses milik *agent* melaui *web service* RESTful. Beberapa teknik digunakan untuk memberi peringkat hasil *query* dan menggunkaan pola yang ditemukan dalam dokumen web semantik (Piraviperumal et al, 2013).