RANCANG BANGUN SKEMA *BACKEND* DENGAN PENERAPANNYA PADA APLIKASI TEKNOLOGI PERIKANAN MODERN DENGAN FITUR *MULTI USER* DAN INVENTARISASI BERBASIS *MULTI PLATFORM*

Proposal Skripsi

Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer

Mencerdaskan dan
Memartabatkan Bangsa

Oleh:
Akbar Maulana Alfatih
1313619003

PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA

2022

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah SWT, karena dengan rahmat dan karunia-Nya, penulis dapat menyelesaikan proposal skripsi yang berjudul **"Perancangan Arsitektur *Search Engine* dengan Mengintegrasikan *Web Crawler*, Algoritma *Page Ranking*, dan *Document Ranking*"**.

Keberhasilan dalam menyusun proposal skripsi ini tidak lepas dari bantuan berbagai pihak yang mana dengan tulus dan ikhlas memberikan masukan guna sempurnanya proposal skripsi ini. Oleh karena itu dalam kesempatan ini, dengan kerendahan hati penulis mengucapkan banyak terima kasih kepada:

1. Yth. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta.
2. Yth. Ibu Ir. Fariani Hermin Indiyah, M.T selaku Koordinator Program Studi Ilmu Komputer.
3. Yth. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
4. Yth. Bapak Drs. Mulyono, M.Kom selaku Dosen Pembimbing II yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap proposal skripsi ini.
5. Ayah dan Ibu penulis yang selama ini telah mendukung dan membantu menyelesaikan proposal skripsi ini.
6. Teman-teman Program Studi Ilmu Komputer 2018 yang telah mendukung dan membantu proposal skripsi ini.

Penulis menyadari bahwa penyusunan proposal skripsi ini masih jauh dari sempurna karena keterbatasan ilmu dan pengalaman yang dimiliki. Oleh karenanya, kritik dan saran yang bersifat membangun akan penulis terima dengan senang hati. Akhir kata, penulis berharap tugas akhir ini bermanfaat bagi semua pihak khususnya penulis sendiri. Semoga Allah SWT senantiasa membalas kebaikan semua pihak yang telah membantu penulis dalam menyelesaikan proposal skripsi ini.

Jakarta, 6 Juni 2022

Lazuardy Khatulistiwa

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I

PENDAHULUAN

A. Latar Belakang Masalah

Saat ini internet merupakan suatu kebutuhan bagi masyarakat secara umum yang berimplikasi terhadap jumlah situs yang terdapat pada *World Wide Web*. Berdasarkan survei yang dilakukan NetCraft pada bulan April tahun 2022, terdapat 1 miliar situs yang terdapat di web dengan 202 juta situs yang aktif. Dengan banyaknya situs tersebut, maka pengguna internet memerlukan sebuah mesin pencari.

Mesin pencari atau yang biasa disebut *Search Engine* merupakan sebuah program komputer yang berguna untuk membantu pengguna dalam mencari situs web berdasarkan permintaan pencarian pengguna. Mesin pencari sebenarnya tidak berbeda dengan *website* pada umumnya, hanya saja perannya lebih terfokus pada pengumpulan dan pengorganisasian berbagai informasi di internet sesuai dengan kebutuhan penggunanya. Selain untuk memudahkan pencarian, mesin pencari juga berguna untuk meningkatkan pengunjung sebuah situs web.

Mesin pencari pertama kali diperkenalkan oleh Alan Emtage pada tahun 1990 dengan nama *Archie*. *Archie* melakukan pengindeksan *file* pada web dan hanya dapat menampilkan daftar nama situs, tidak dengan isi atau kontennya (Seymour dkk., 2011). Setelah kemunculan *Archie*, di tahun berikutnya mulai bermunculan mesin pencari baru. Pada tahun 1993 *Aliweb* muncul dengan memberikan kesempatan pengguna untuk mengunggah halaman situsnya agar terindeks dan memberikan deskripsi untuk halaman tersebut. Kemudian *Altavista* muncul pada tahun 1995 yang menggunakan teknik dan algoritma yang lebih maju. Setelah

Altavista, pada 2004 muncullah *Yahoo* sebagai salah satu mesin pencari yang maju di mana pengelola dan pemilik situs dapat menambah dan mengedit informasi dalam web tanpa mengeluarkan biaya. Namun, perkembangan mesin pencarian *Yahoo* sangat lambat sehingga pengguna internet perlahan mulai mencari mesin pencari yang lain (Seymour dkk., 2011).

Search engine yang saat ini paling banyak digunakan adalah *Google*. Mesin pencari yang diluncurkan pada tahun 1998 ini berkembang jauh berbeda dengan mesin pencari yang lain. Mereka mampu mengindeks halaman-halaman dan melakukan perangkingan, yang berarti bahwa halaman yang paling sering dikunjungi atau diklik akan berada paling atas dalam pencarian dengan *keyword* terkait.

Search engine dalam perkembangannya tidak hanya memuat halaman berisi informasi, namun juga dapat menjadi sarana iklan. Pada mesin pencari *Google*, terdapat layanan iklan yang disebut dengan *Adwords*. Layanan ini memungkinkan untuk pemilik usaha untuk mengiklankan situs web usahanya, sehingga saat pengguna Google mencari suatu barang, hasil pencarian *Google* akan menampilkan situs web pemilik usaha di bagian teratas dan tertulis sebagai "Ad" atau iklan.

Dalam arsitektur pembuatan *search engine*, terdapat salah satu bagian yang disebut *crawler*. *Crawler* berfungsi untuk mengumpulkan data-data yang akhirnya data-data tersebut akan diproses dan ditampilkan ke pengguna. *Crawler* ini perlu dirancang sebaik mungkin untuk mengumpulkan data yang dapat mendukung *search engine*.

Web crawler pertama kali dimunculkan pada tahun 1944 bernama *RBSE* dengan dasar dua program yaitu *Spider* dan *Mite* (Eichmann, 1994). Dua program dasar ini berguna untuk membentuk antrian data dalam *database* serta untuk mengunduh halaman dari sebuah web. Guna dari *web crawler* sendiri adalah sebagai

perangkat lunak yang secara otomatis bekerja untuk menjelajahi *website* dengan cara terorganisir.

Web crawler berperan penting pada *search engine* karena berfungsi untuk mengumpulkan data sebanyak-banyaknya. Tanpa *web crawler*, *search engine* tidak akan mempunyai data dan tidak bisa melakukan *indexing*. Data yang dihasilkan dari sebuah *web crawler* ini adalah data yang terbaru dan memiliki nilai keakuratan yang tinggi. *Web crawler* yang dimiliki *Google* bernama *Googlebot*, *web crawler Bing* bernama *Bingbot*, dan *web crawler Yahoo* bernama *Slurp bot*.

Dalam jurnal tahun 2020 oleh Kaur dan Geetha, mereka membuat sebuah *web crawler* untuk *hidden web* menggunakan *SIM+HASH* disertai dengan *Redis Server* (Kaur dan Geetha, 2020). *Web crawler* ini mampu menghasilkan hasil halaman dengan tingkat kecocokan yang maksimum. Meski hasil yang ditampilkan sangat relevan dengan *keyword* yang dimasukkan, namun dalam pembangunannya, *web crawler* ini sangatlah rumit meski komponennya hampir sama dengan *web crawler* pada umumnya.

Penelitian yang dilakukan Muhammad Fathan menghasilkan *web crawler* yang dapat berjalan dengan baik di situs yang menggunakan *html* versi 4 maupun *html* versi 5. Algoritma *crawler* yang digunakan pada penelitian tersebut adalah algoritma *breadth first search crawling* dan algoritma *modified similarity based crawling* yang dikembangkan oleh Google (Qoriiba, 2021).

Dalam menghasilkan informasi berdasarkan *keyword* yang dimasukkan pengguna pada *search engine*, diperlukan sebuah sistem temu kembali informasi atau yang biasa disebut dengan *Information Retrieval*. Temu kembali informasi ini dimaksudkan untuk menemukan material atau data-data (sekumpulan dokumen) pada penyimpanan yang tak beraturan (biasanya berbentuk teks) di mana menjadi kebutuhan akan sebuah informasi dari kumpulan data berukuran besar (dalam

penyimpanan komputer) (Manning dkk., 2009).

Information retrieval mampu menemukan informasi yang tersimpan dalam sistem sesuai dengan *keyword* tertentu yang dimasukkan pengguna. Teknologi ini mencakup 2 hal, yaitu pembobotan dan perangkingan. Peringkat tertinggi akan diisi oleh data yang paling relevan dengan *keyword* masukan pengguna dan akan terus diurutkan berdasarkan relevansinya.

Google PageRank adalah salah satu algoritma perangkingan situs (*page ranking*) yang berfungsi untuk menentukan situs web mana yang lebih populer. Algoritma ini diperkenalkan oleh Sergey Brin dan Larry Page pada tahun 1998 dalam jurnalnya yang berjudul *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. Konsep dari perangkingan *PageRank* adalah semakin banyak situs lain yang memasang *link* ke situsnya, maka situs tersebut akan semakin populer, dengan anggapan bahwa konten situs tersebut lebih bermanfaat daripada konten situs lain.

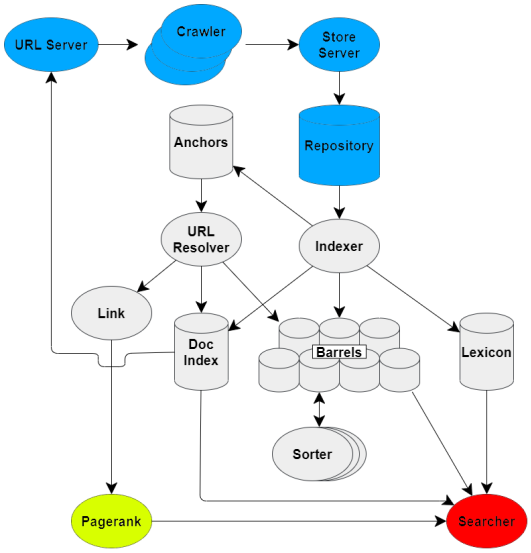
Dalam pengembangan *search engine*, selain perangkingan situs web diperlukan sebuah metode *document ranking*, yaitu untuk memverifikasi apakah konten dalam situs web tersebut relevan atau tidak. Salah satu metode yang paling umum digunakan adalah metode *TF-IDF*. *TF-IDF* atau *Term Frequency-Inversed Document Frequency* adalah suatu metode untuk memberikan bobot kata yang berhubungan terhadap suatu dokumen. Metode ini menggunakan algoritma yang menggabungkan 2 nilai yaitu *Term Frequency (TF)* yang merupakan nilai frekuensi kemunculan kata dan *Inverse Document Frequency (IDF)* yang merupakan nilai untuk mengukur seberapa penting sebuah kata. Dengan menggunakan metode ini pencarian pada *search engine* akan memiliki hasil yang relevan sesuai dengan *keyword* masukan pengguna.

Dari penelitian yang dilakukan oleh Juan Ramos yang berjudul "*Using*

TF-IDF to Determine Word Relevance in Document Queries", terdapat kesimpulan bahwa *TF-IDF* merupakan algoritma sederhana dan efisien yang dapat menghasilkan kumpulan dokumen dengan tingkat kesesuaian yang tinggi sesuai dengan *keyword* yang dimasukkan.

Terdapat beberapa kekurangan dari algoritma TF-IDF, salah satunya adalah algoritma ini tidak bisa mengidentifikasi kata di bahasa inggris sesuai dengan *tense*-nya. Sebagai contoh, algoritma ini akan memperlakukan kata *"play"*, dan *"plays"* sebagai kata yang berbeda. Namun, hal ini dapat diatasi dengan melakukan proses *stemming* sebelum *TF-IDF*, yaitu mengubah kata yang tadinya menggunakan berbagai bentuk tertentu seperti *"play"*, *"plays"*, atau *"played"* menjadi sebuah kata dasar *"play"* (Qaiser dan Ali, 2018).

Search engine yang populer saat ini telah memiliki teknik dan algoritma yang lebih maju dan berbeda-beda. Perusahaan-perusahaan besar pemiliknya memiliki tekniknya masing-masing dan enggan membeberkan detail algoritma yang ia pakai dalam mengembangkan *search engine* karena bersifat komersial. Oleh karena itu, pengembangan *search engine* ini menarik untuk dilakukan.



Gambar 1.1: *High Level Google Architecture* (Brin dan Page, 1998)

Penelitian ini akan merancang dan membangun *search engine* berdasarkan arsitekturnya dengan mengintegrasikan penelitian dari Muhammad Fathan yang berfokus pada perancangan *crawler* menggunakan algoritma *modified similarity based crawling* yang digambarkan sesuai arsitektur pada gambar 1.1 dengan arsiran berwarna biru, penelitian Mohammad Riza yang mengimplementasikan algoritma perangkingan *Google PageRank* yang terdapat pada arsiran kuning, dan penelitian Savira Rahmayanti yang mengembangkan sistem pencarian teks menggunakan metode *TF-IDF* yang terdapat pada arsiran merah.

B. Rumusan Masalah

Dari uraian latar belakang di atas, perumusan masalah pada penelitian ini ialah "Bagaimana perancangan arsitektur *search engine* berbasis terminal menggunakan *web crawler*, *page ranking*, dan *document ranking* berbasis *TF IDF*?"

C. Pembatasan Masalah

Pembatasan masalah pada penelitian ini antara lain:

1. Situs yang menjadi titik awal saat proses *crawling* adalah indosport.com dan curiouscuisiniere.com.
2. Proses *crawling* dilakukan dengan jangka waktu 1 hari.
3. Mengkombinasikan metode relevansi pencarian berbasis *TF IDF* dan *Google PageRank*

D. Tujuan Penelitian

1. Membuat search engine berbasis terminal yang akan menerima *keyword* pencarian dari pengguna menggunakan *crawler*, *Google PageRank*, dan

pencarian teks berbasis *TF IDF*.

2. Membuat *API* dari *REST search engine* yang akan dibangun.

E. Manfaat Penelitian

1. Bagi penulis

   Memperluas pengetahuan tentang *search engine*, menambah pengalaman dalam *programming*, memperoleh gelar sarjana di bidang Ilmu Komputer, serta menjadi media untuk penulis dalam mengaplikasikan ilmu yang didapatkan dari kampus.

2. Bagi Program Studi Ilmu Komputer

   Penelitian ini dapat menjadi pembuka untuk penelitian di masa depan, dan dapat memberikan panduan bagi mahasiswa program studi Ilmu Komputer tentang rancang bangun aplikasi *search engine*.

3. Bagi Universitas Negeri Jakarta

   Menjadi evaluasi akademik program studi Ilmu Komputer dalam penyusunan skripsi sehingga dapat meningkatkan kualitas pendidikan dan lulusan program studi Ilmu Komputer di Universitas Negeri Jakarta.

BAB II

KAJIAN PUSTAKA

A. Pengertian *Search Engine*

Search engine adalah sebuah mesin atau program yang mampu menampilkan banyak informasi yang relevan terkait dengan *keyword* masukan pengguna. Sistem ini adalah sebuah sistem yang mengindeks, mengumpulkan segala informasi yang tersimpan dalam internet, menyaringnya berdasarkan *keyword input*, dan menampilkan halaman *website* terkait dengan perangkingan guna memudahkan dalam mendapatkan informasi.

Secara singkat, *search engine* juga dapat disebut sebagai sebuah navigasi atau penunjuk dalam melakukan *surfing* dalam dunia internet. Dengan memanfaatkan *search engine*, kita dengan mudah akan mendapatkan informasi yang diperlukan dengan rekomendasi halaman yang paling terkenal berada dalam ranking paling atas.

B. Sejarah *Search Engine*

Tahun 1990 adalah tahun pertama kemunculan *search engine* untuk pertama kali dan diberi nama dengan *Archive* atau *Archie*. Dibangun oleh 3 mahasiswa Ilmu Komputer dari Universitas Mc Gill di Kanada bernama J.Peter Deutsch, Bill Heelan, dan Alan Emtage. *Archie* lahir dari sebuah *database* yang berisi *file* dari ratusan sistem. Cara kerja *Archie* tidak dengan mengindeks isi konten dari sebuah web, namun dengan cara menjangkau semua FTP dan membuat daftar *file* untuk membuat indeks pencarian. Untuk dapat menggunakan *Archie* dalam melakukan pencarian, diperlukan pengetahuan yang cukup tentang *UNIX* karena *Archie* dibangun menggunakan perintah-perintah *UNIX* (Seymour dkk., 2011).

Satu tahun setelah *Archie* diluncurkan, muncullah *Gopher* yang membawa dua *search engine* yang disebut *Jughead* dan *Veronice*. Dua program pencarian ini bekerja mirip *Archie*, yakni mengumpulkan nama judul dan *file* dan diindeks ke dalam *Gopher*. Dibuat oleh Mark McCahill, *Gopher* diharapkan dapat mencari, mengambil, dan menyebarkan *file* melalui internet dengan tingkat penyimpanan informasi lebih kuat. Namun disayangkan, *Gopher* memiliki beberapa fitur yang tidak mampu didukung oleh *website* pada tahun 1991.

Pada tahun 1993, sebuah *search engine* bernama *Aliweb* telah rilis. Untuk melakukan pengindeksan, pengguna *Aliweb* harus mendaftarkan situsnya ke *Aliweb Server* (Seymour dkk., 2011). Hal ini dikarenakan *Aliweb* tidak mengindeks situs secara otomatis. Pada akhir tahun 1993, perilisan *search engine* lain yang diberi nama *Jumpstation* dilakukan. Dengan menggunakan *web robot*, *Jumpstation* mencari halaman web lalu mengindeksnya. Hal ini membuat *Jumpstation* menjadi *search engine* pertama yang memenuhi 3 fitur utama sebuah *search engine* yakni *searching*, *indexing*, dan *crawling*. Meski masih terbatas dalam pengindeksan, namun *Jumpstation* sanggup memberikan fasilitas pengindeksan *headings* dan judul yang ditemukan di web.

Tahun 1994 pada bulan April, Brian Pinkerton dari Universitas Washington meluncurkan *WebCrawler* yang menjadi *search engine* pertama yang menawarkan pencarian teks secara lengkap di mana pengguna mampu mencari kata yang terdapat pada halaman web. Pengguna dapat memasukkan kata apapun yang mereka kira berhubungan dengan apa yang mereka cari dan *Webcrawler* akan menampilkan setiap web yang memiliki kata tersebut. Hal ini di kemudian hari menjadi standar dari sebuah *search engine*.

Setelah *WebCrawler* bermunculan banyak *search engine* serupa yang berusaha menyaingi demi popularitas. Beberapa *search engine* yang bermunculan di

antaranya adalah *AltaVista*, *Infoseek*, *Inktomi*, dan *Nocthern Light*. *Yahoo!* dianggap salah satu yang terfavorit tetapi karena fungsi pencariannya hanya berada di direktori webnya, *Yahoo!* membuat *user* kurang nyaman karena teks tidak disalin dari *website*.

Selanjutnya pada tahun 1995, *AltaVista* menunjukkan kepopulerannya. Pengindeksan *AltaVista* dibuat oleh Michael Burrows dan crawlernya dikerjakan oleh Louise Monier. *Server AltaVista* menjadi *server* paling kuat yang mampu menangani jutaan hit dalam sehari, dan membuat *AltaVista* menjadi *search engine* paling cepat. Hal yang menyebabkan *AltaVista* menjadi populer adalah adanya fitur yang mampu memunculkan informasi yang sesuai seperti yang dimasukkan dari *user keyword*.

Ask Jeeves (Ask) adalah *search engine* yang dibuat oleh David Warthen dan Garrett Gruener dan didirikan tahun 1996 di Barkeley, California (Seymour et al., 2011). Ide awal *Ask* muncul supaya memudahkan pengguna menemukan jawaban dari pertanyaan yang ditanyakan menggunakan bahasa yang umum digunakan. Saat ini *Ask.com* masih tersedia dan terdapat fitur tambahan, yaitu dapat menerima pertanyaan matematika, kamus, dan pertanyaan konversi.

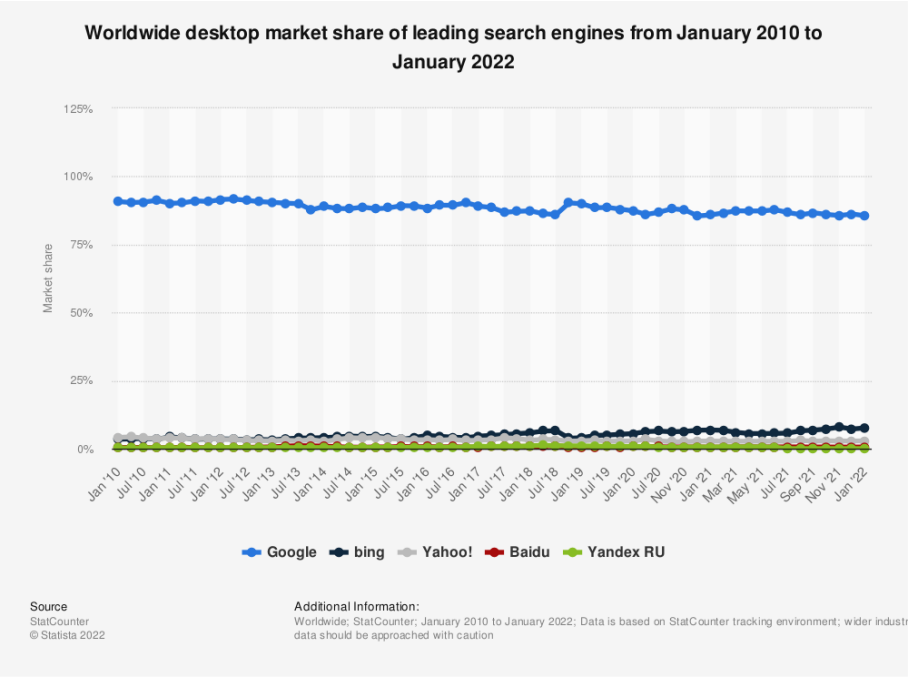
Dan pada tahun 1998, lahir satu *search engine* yang di kemudian hari menjadi *search engine* yang paling terkenal di seluruh negara. *Google* diciptakan oleh Sergey Brin dan Larry Page yang merupakan mahasiswa Stanford California. Cara mereka untuk meningkatkan kualitas pencarian agar lebih akurat adalah dengan menambahkan fitur *PageRank* yang di mana *PageRank* adalah metode untuk menghitung peringkat setiap halaman web (Page dkk., 1999).

Google search engine telah meminimalisir hasil penelusuran yang tidak berguna di hasil penelusuran teratas, sehingga memudahkan pengguna untuk mencari informasi yang ada di web. Mulai dari tahun 2000, *Google search engine*

menjadi terkenal hingga saat ini.

Di awal abad ke 20 tepatnya pada tanggal 18 Januari 2000, muncul *search engine* bernama *Baidu*. *Baidu* berpusat di Beijing, RRC. *Baidu* merupakan *search engine* untuk *website*, *file audio* dan gambar berbahasa Cina dan Jepang. *Baidu* merupakan perusahaan Cina yang masuk dalam indeks NASDAQ-100 pertama kali. Robin Li dan Eric Xu merupakan pendiri perusahaan *Baidu*.

MSN search diluncurkan pertama kali oleh Microsoft pada tahun 1998, menggunakan *Inktomi* sebagai hasil penelusurannya (Seymour dkk., 2011). Microsoft memiliki *web crawler* sendiri bernama *msnbot*, dan perlahan memulai menciptakan teknologi *search engine* sendiri di tahun 2004 yang bernama *Bing*. Bing resmi muncul pada tahun 2009 di awal bulan Juni. Kemudian satu bulan lebih setelahnya tepatnya tanggal 29 Juli 2009, Microsoft dan Yahoo! membuat kesepakatan di mana teknologi search engine *Bing* akan mendukung hasil pencarian *Yahoo! search*.

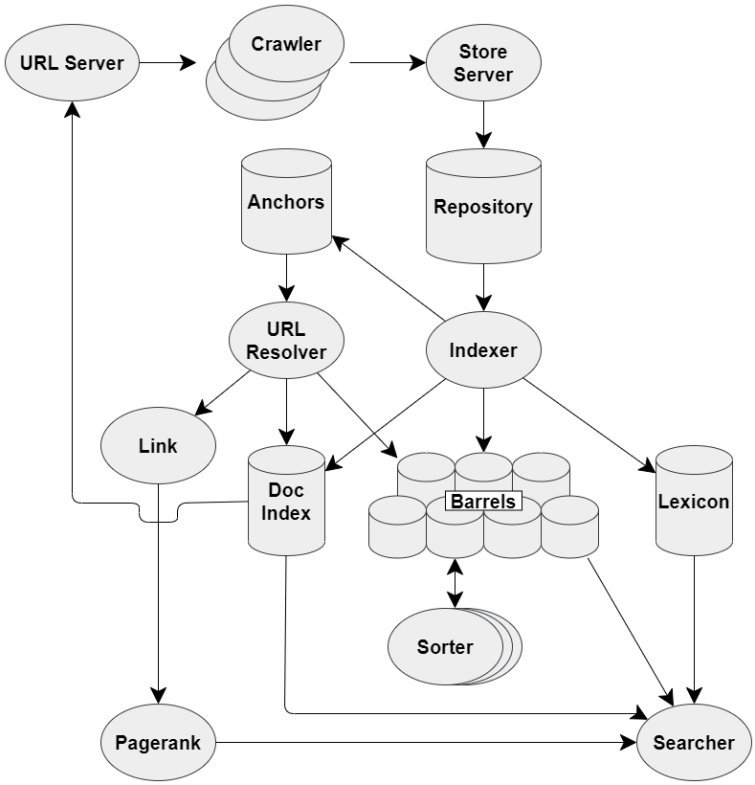


Gambar 2.1: *Market share search engine* tahun 2010 hingga 2022 (Statista, 2022)

Berdasarkan data pada gambar 2.1, *search engine* yang paling banyak digunakan dari tahun 2010 hingga 2022 adalah *Google*. Di peringkat kedua ada *Yahoo* dan peringkat ketiga adalah *Bing*. Selama 12 tahun *Google* selalu berada di peringkat pertama dengan persentase di atas 85%. Sedangkan *Bing* dan *Yahoo* pada bulan Januari 2022 memiliki persentase 7.61% dan 2.85%.

C. Arsitektur *Search Engine*

Cara kerja *search engine* adalah menyimpan semua informasi dari berbagai *website*. Informasi dari halaman-halaman tersebut diekstrak menggunakan sebuah program yang disebut *Web Crawler*, lalu konten di setiap halaman akan dianalisis untuk menentukan bagaimana halaman diindeks. Gambar 2.2 merupakan *High Level Google Architecture* (Brin dan Page, 1998).



Gambar 2.2: *High Level Google Architecture* (Brin dan Page, 1998)

Dari gambar 2.2 dapat dijelaskan secara berurutan sebagai berikut:

1. *Server* mengirimkan *URL* untuk dikumpulkan oleh crawler.
2. Pengunduhan halaman *dari URL tersebut* dijalankan oleh beberapa *crawler*.
3. Halaman web yang telah diunduh atau diekstrak dikirim ke *store server*.
4. Dari *store server* berpindah untuk disimpan ke dalam *repository* di mana setiap halaman web memiliki sebuah ID yang disebut *docID*.
5. Bagian *indexer* dan *sorter* melakukan pengindeksan.
6. Setiap dokumennya dirubah menjadi sekumpulan kata yang disebut *hits*.
7. *Indexer* menyalurkan *hits* ke dalam satu set *barrels* dan membuat *forward index* yang diurutkan sebagian.
8. *Indexer* menguraikan *URL* di setiap halaman web yang tersimpan dan menyaring informasi untuk disimpan ke dalam *anchor file*.
9. *URL resolver* mendeteksi isi *ancor file* dan merubah *URL* yang sebelumnya relatif menjadi absolut, serta merubahnya menjadi sebuah docID.
10. Menaruh *anchor text* ke *forward index* terkait *docID* yang telah ditunjukkan *anchor*.
11. Membuat *database* berisi link yang berpasangan dengan *docID* untuk penghitungan *PageRank*.
12. *Sorter* mengambil *barrels* yang diurutkan berdasarkan *docID* dan *wordID* untuk menghasilkan *inverted index*.
13. Program *DumpLexicon* mengambil daftar wordID dan menghasilkan *lexicon* baru untuk dipakai oleh *searcher* bersama dengan lexicon yang dibuat *indexer*.

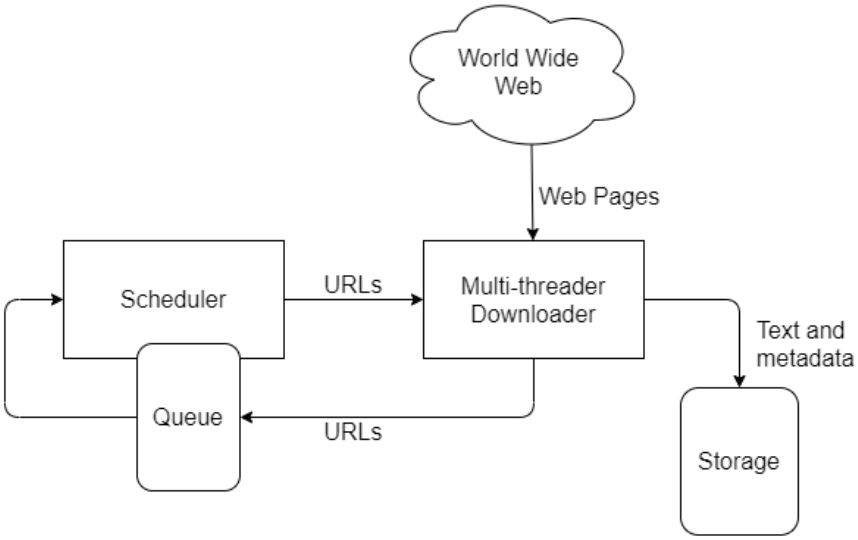
14. *Searcher* dilakukan oleh *web server* dan memakai *lexicon* yang dibuat oleh *DumpLexicon* bersama dengan *inverted index* dan *PageRank* untuk menjawab pertanyaan.

Saat mengimplementasikan proses ini pada tahun 1998, Google membutuhkan waktu kurang lebih 9 hari untuk mengunduh data dari 26 juta halaman web, yang berarti prosesnya membutuhkan waktu sekitar 33,44 halaman per detik. Proses ini berjalan secara berulang untuk mendapatkan perubahan informasi atau data yang baru dari seluruh halaman web. Hampir seluruh proses diimplementasikan menggunakan bahasa *C* atau *C++*, sisanya seperti *crawler* dan *URL server* menggunakan bahasa *Python*.

D. *Web Crawler*

Web Crawler adalah program yang mengambil halaman web, biasanya untuk digunakan oleh mesin pencari (Pinkerton, 1994). Secara kasar, *crawler* dimulai dari *URL* halaman awal P_0. Kemudian *crawler* merayap ke P_0, mengekstrak semua *URL* yang terdapat di dalam P_0, dan menambahkannya ke antrian *URL* untuk dipindai. Setelah itu, *crawler* mengambil *URL* dari antrian (dengan urutan tertentu), dan mengulangi prosesnya (Cho dkk., 1998).

E. Arsitektur *Web Crawler*



Gambar 2.3: *High Level Architecture of Web Crawler* (Castillo, 2005)

Web crawler adalah bagian penting dari *search engine*. *Crawler* membutuhkan strategi *crawling* yang baik dan arsitektur yang sangat optimal. Desain sistem harus dioptimalkan untuk membentuk sistem berkinerja tinggi yang dapat mengunduh ratusan juta halaman selama beberapa minggu. *High level architecture of Web crawler* ditunjukkan pada Gambar 2.3, membutuhkan *scheduler* dan *multi-threader downloader*. Dua struktur data utama adalah *Web page* (*text*) dan *URL queue*.

F. Algoritma *Crawling*

Secara umum, algoritma yang digunakan untuk *crawling* menggunakan algoritma *breadth first search*. Algoritma ini merupakan bentuk paling sederhana dari algoritma *crawling*. Idenya adalah *crawling* dimulai dari sebuah *link*, kemudian terus mengikuti *link* lain yang terhubung. Algoritma *typical crawling model* dapat dilihat pada algoritma 1 (Castillo, 2005).

Algorithm 1 *Typical Crawling Model* (Castillo, 2005)

Require: $p_1, p_2, ..., p_n$ starting URLs

1: $Q = p_1, p_2, ..., p_n$, queue of URLs to visit.
2: $V = \emptyset$, visited URLs.
3: **while** $Q \neq \emptyset$ **do**
4: Dequeue $p \in Q$, select p according to some criteria.
5: Do an asynchronous network fetch for p.
6: $V = V \cup \{p\}$
7: Parse p to extract text and extract outgoing links
8: $\Gamma^+(p) \leftarrow$ pages pointed by p
9: **for** each $p' \in \Gamma^+(p)$ **do**
10: **if** $p' \notin V \wedge p' \notin Q$ **then**
11: $Q = Q \cup \{p'\}$
12: **end if**
13: **end for**
14: **end while**

Algoritma ini dimulai dengan mengumpulkan *URL page* $p_1, p_2,$ hingga p_n ke dalam variabel Q. Kemudian, page tersebut dilakukan proses *fetch* satu per satu dan mengambil teks serta *outgoing link* dari *page*. *Outgoing link* yang didapat akan dimasukkan ke dalam variabel Q jika link tersebut belum ada di variabel Q dan belum masuk dalam proses *fetch*. Proses ini akan berulang sampai *page* pada variabel Q kosong.

Karena sumber daya dan waktu yang terbatas, *crawler* harus menentukan urutan *URL* yang akan diproses terlebih dahulu. Algoritma *modified similarity-based crawling* adalah algoritma untuk mengatasi masalah tersebut. *Modified similarity-based crawling algorithm* dapat dilihat pada algoritma 2 (Cho dkk., 1998).

Algorithm 2 *Modified similarity-based crawling* (Cho dkk., 1998)

```
1: enqueue(url_queue, starting_url);
2: while (not empty(hot_queue)) and not empty(url_queue) do
3:     url = dequeue2(hot_queue, url_queue);
4:     page = crawl_page(url);
5:     if [page contains 10 or more computer in body or one computer in title] then
6:         hot[url] = TRUE;
7:     end if
8:     enqueue(crawled_pages, (url, pages));
9:     url_list = extract_urls(page);
10:    for each u in url_list do
11:        enqueue(links, (url, u));
12:        if [u not in url_queue] and [u not in hot_queue] and [(u,-) not in
       crawled_pages] then
13:            if [u contains computer in anchor or url] then
14:                enqueue(hot_queue, u);
15:            else if [distance_from_hotpage(u) < 3] then
16:                enqueue(hot_queue, u);
17:            else
18:                enqueue(url_queue, u);
19:            end if
20:        end if
21:        reorder_queue(url_queue);
22:        reorder_queue(hot_queue);
23:    end for
24: end while
```

Terdapat perbedaan pada algoritma 1 *typical crawling model* dan algoritma 2 *Modified similarity-based crawling*. Pada algoritma 2 *starting_url* dimasukkan ke *url_queue*. Selain *url_queue*, ada juga *hot_queue*. *hot_queue* merupakan kumpulan *URL hot page*. Suatu *page* dikatakan *hot*, karena memiliki salah satu dari kriteria berikut:

1. Page yang berisi lebih dari 10 kata spesifik (dicontohkan sebagai kata "*computer*") pada bagian *body* atau 1 kata *computer* di bagian *title*.
2. Kata spesifik (*computer*) tersebut terdapat pada *anchor* atau *URL*.
3. *URL* yang memiliki hubungan dengan *hot_page*, masuk ke dalam kumpulan *URL* di *hot_queue*, jika jaraknya kurang dari 3 *link/node*.

Kumpulan *URL* baru yang didapat pada proses *crawling* akan dimasukkan ke dalam *hot_queue* jika URL tersebut termasuk *hot page*, jika tidak maka akan dimasukkan ke dalam *url_queue*. Lalu *hot_queue* dan *url_queue* akan diurutkan lagi pada fungsi *reorder_queue* dengan menggunakan *backlink count*. *URL* yang mempunyai nilai *backlink count* tertinggi akan menempati urutan teratas dan akan diproses terlebih dahulu. Nilai *backlink count* diperoleh dari banyaknya *URL* yang ditempatkan pada *URL* lain.

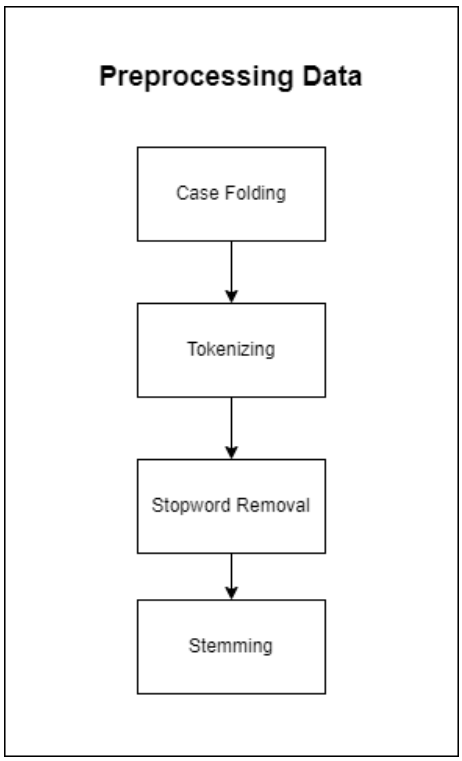
G. *Preprocessing Data*

Setiap dokumen atau data seringkali memiliki struktur yang berubah-ubah atau tidak terstruktur. Oleh karena itu, diperlukan suatu proses yang dapat mengubah bentuk data yang sebelumnya tidak terstruktur menjadi data terstruktur. Proses konversi ini disebut *text processing* (Feldman dan Sanger, 2007).

Data teks tersebut berisi banyak variasi dari mulai huruf hingga tanda baca. Variasi huruf harus konsisten, yaitu hanya menggunakan huruf besar saja atau huruf

kecil saja. Selain itu, tanda baca juga perlu dihilangkan untuk mengurangi *noise* selama pencarian informasi.

Preprocessing data dilakukan agar data yang digunakan bebas noise, memiliki ukuran yang lebih kecil, dan lebih terorganisir sehingga dapat diproses lebih lanjut. Ada beberapa proses dalam tahap *preprocessing data* yaitu *case folding*, *tokenizing*, *stop word removal* dan *stemming* seperti pada gambar 2.4.



Gambar 2.4: Tahapan *preprocessing data*

Proses pertama pada tahap *preprocessing* adalah *case folding*. Tujuan dari proses case folding adalah untuk menyamakan karakter pada data. Proses ini mengubah seluruh huruf pada data menjadi huruf kecil. Karakter-karakter huruf besar dari A sampai Z yang terdapat pada data diubah kedalam karakter huruf kecil a sampai z. Sedangkan karakter-karakter selain huruf seperti tanda baca dan angka akan dihilangkan dari data dan dianggap sebagai *delimeter* atau batas pemisah

(Manning dkk., 2009).

Sebelum data teks dapat diproses lebih lanjut, data yang tadinya terdiri dari banyak kalimat harus dipecah menjadi kata-kata terpisah, yang mana proses ini disebut tokenization. Strategi umum yang dilakukan selama proses tokenization adalah memisahkan kata dengan *whitespace* sebagai acuan pembatasnya dan menghilangkan tanda baca. Potongan-potongan kata yang sudah dihasilkan disebut dengan *token*.

Tahap selanjutnya setelah *tokenizing* adalah *stop word removal* dan *stemming*. *Stop word removal* adalah pengambilan kata-kata penting dengan dari *token* dengan cara membuang kata yang kurang penting atau tidak bermakna. Contoh *stop word* dalam bahasa Indonesia dan akan dibuang dalam proses ini adalah "yang", "dan", "di", dll. Sedangkan *stemming* adalah proses penghilangan imbuhan pada sebuah kata sehingga menjadi kata dasar.

H. Pembobotan *TF-IDF*

Metode *Term Frequency-Inverse Document Frequency TF-IDF* adalah salah satu metode yang paling umum digunakan untuk menghitung bobot setiap kata dalam pencarian informasi. Cara ini juga dinilai efisien, sederhana dan memiliki hasil yang akurat.

TF-IDF adalah metode pemberian bobot pada hubungan antara kata (*word*) dan dokumen. *TF-IDF* adalah ukuran statistik yang digunakan untuk mengevaluasi pentingnya sebuah kata dalam dokumen atau kumpulan kata. Frekuensi kata dalam dokumen tertentu menunjukkan pentingnya dalam dokumen. Frekuensi dokumen yang berisi kata tersebut menunjukkan seberapa umum kata tersebut. Jika sering muncul dalam satu dokumen, bobot kata lebih besar, dan jika muncul di banyak dokumen, bobot kata akan lebih kecil.

Term Frequency (TF) menghitung berapa kali sebuah kata muncul dalam dokumen. Karena panjang setiap dokumen bisa berbeda, nilai *TF* dibagi dengan panjang dokumen (jumlah kata dalam dokumen). Berikut rumus dari *Term Frequency (TF)*:

$$tf_{t,d} = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ document} \tag{2.1}$$

Di mana maksud dari pembilang pada persamaan di atas adalah jumlah kata t muncul dalam satu dokumen dan maksud dari penyebutnya adalah jumlah semua kata yang ada dalam satu dokumen tersebut.

Setelah menghitung *Term Frequency (TF)*, selanjutnya adalah menghitung nilai *Inverse Document Frequency (IDF)*, yang merupakan ukuran seberapa penting suatu kata. *IDF* akan memberikan nilai pada kata-kata yang biasanya muncul sebagai kata yang kurang penting berdasarkan kemunculannya di seluruh dokumen. Semakin kecil nilai IDF, semakin tidak penting kata tersebut, dan sebaliknya. Berikut rumus dari *Inverse Document Frequency (IDF)*:

$$idf_t = log(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it}) \tag{2.2}$$

Di mana maksud dari pembilang pada persamaan di atas adalah jumlah dokumen yang ada pada dataset dan maksud dari penyebutnya adalah jumlah dokumen pada dataset yang terdapat kata t di dalamnya.

Nilai idf dari kata yang kemunculannya jarang akan bernilai tinggi, sedangkan nilai idf dari kata yang kemunculannya sering nilainya akan cenderung rendah (Manning dkk., 2009).

Setelah mempunyai nilai *TF* dan *IDF*, perhitungan bobot *TF IDF* adalah hasil

perkalian dari nilai *TF* dan *IDF*.

$$tfidf_{t,d} = tf_{t,d} \times idf_t \tag{2.3}$$

I. *Google PageRank*

Setiap kali melakukan pencarian di *Google*, terdapat hasil *website* yang muncul di halaman secara berurutan, sebagian itu adalah karena *Google PageRank*. Sederhananya, *PageRank* menganalisis jumlah tautan masuk ke setiap halaman di web, dan menetapkan setiap halaman sebuah peringkat yang ditentukan oleh peringkat setiap halaman yang menautkannya. *PageRank* mengikuti model probabilistik dari seorang peselancar web, yang mengunjungi halaman tertentu, dan kemudian mengklik link di halaman itu secara acak. Intinya, peringkat setiap halaman mewakili kemungkinan bahwa setiap peselancar web akan mendarat di sana. Meskipun secara konseptual sederhana, *PageRank* secara luas dianggap sebagai salah satu algoritma terpenting yang pernah ditemukan di industri teknologi.

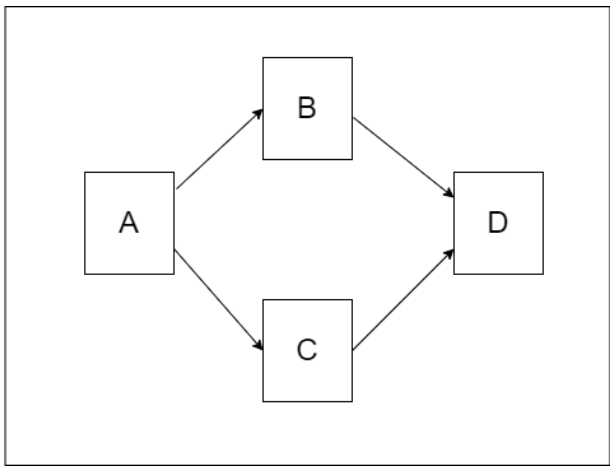
Meskipun *PageRank* secara populer dikaitkan dengan Larry Page dan Sergey Brin, pendiri *Google*, versi awal masalah *eigenvalue* dieksplorasi pada tahun 1976 oleh Gabriel Pinski dan Francis Narin dalam upaya mereka untuk menentukan peringkat jurnal ilmiah. Thomas Saaty mengeksplorasi masalah yang sama pada tahun 1977 dalam konsepnya tentang *Analytic Heirarchy Process* yang menimbang keputusan, dan Bradley Love dan Steven Sloman menggunakannya pada tahun 1995 sebagai model kognitif untuk konsepnya juga.

Namun, baru pada tahun 1996 Larry Page dan Sergey Brin mengadopsi algoritma ini untuk digunakan dalam mesin pencari. Sebagai bagian dari proyek penelitian Stanford tentang penerapan mesin pencari jenis baru, Sergey Brin mengkonseptualisasikan gagasan untuk memeringkat halaman web dengan jumlah

tautan yang mengarah ke sana. Makalah pertama mereka tentang masalah ini diterbitkan pada tahun 1998, yang mencakup prototipe awal mesin pencari Google, dan Page dan Brin juga mematenkan prosesnya (Universitas Stanford memegang paten, tetapi memberikan hak lisensi eksklusif kepada Google dengan imbalan 1,8 juta saham darinya). Sejak saat itu, algoritma ini tetap menjadi komponen integral dari mesin telusur *Google* selama bertahun-tahun, meskipun ini bukan satu-satunya kriteria yang digunakan *Google* untuk menentukan peringkat hasil penelusurannya.

Sebagai algoritma yang berlaku untuk *graph* apa pun secara umum, *PageRank* telah digunakan di banyak aplikasi lain selain mesin pencari, termasuk analisis jaringan jalan, ilmu saraf, bibliometrik, sastra, dan bahkan olahraga. Dalam industri teknologi, varian dari algoritma *PageRank* tetap digunakan sampai sekarang; *Twitter* menggunakannya untuk menyarankan pengguna untuk mengikuti, dan *Facebook* mengimplementasikannya di belakang layar sebagai bagian dari algoritma teman yang disarankan.

Algoritma *PageRank*, salah satu yang paling banyak digunakan terkait algoritma peringkat halaman, menyatakan bahwa jika suatu halaman memiliki *link* penting ke sana, link ke halaman lain juga menjadi penting. Oleh karena itu, *PageRank* memperhitungkan *backlink* dan menyebarkan peringkat melalui tautan, yaitu sebuah halaman memiliki peringkat tinggi jika jumlah peringkat *backlink*-nya tinggi. Gambar 2.5 menunjukkan contoh dari *backlink*: halaman A adalah *backlink* halaman B dan halaman C sedangkan halaman B dan halaman C adalah *backlink* dari halaman D.



Gambar 2.5: Contoh *backlink*

1. *Simplified PageRank*

Versi *PageRank* yang sedikit disederhanakan didefinisikan sebagai (Page dkk., 1999):

$$PR(u) = c \sum_{v \in B(u)} \frac{PR(v)}{N_v} \tag{2.4}$$

Di mana u merepresentasikan halaman web. $B(u)$ adalah kumpulan halaman yang merujuk ke u. $PR(u)$ dan $PR(v)$ masing-masing merupakan skor peringkat halaman u dan v. N_v menunjukkan jumlah tautan keluar dari halaman v. Sedangkan c adalah konstanta yang digunakan untuk normalisasi dan biasanya bernilai 0.85 namun dapat diubah sesuai keinginan.

Di *PageRank*, skor peringkat halaman, p, dibagi rata di antara tautan keluarnya. Nilai yang ditetapkan ke tautan keluar halaman p selanjutnya digunakan untuk menghitung peringkat halaman yang ditunjuk oleh halaman p. Skor peringkat halaman situs web dapat dihitung secara iteratif mulai dari halaman web mana pun. Dalam sebuah situs web, dua atau lebih halaman mungkin terhubung satu sama lain untuk membentuk satu lingkaran. Jika halaman ini tidak merujuk tetapi dirujuk oleh

halaman web lain di luar *loop*, mereka akan mengakumulasi peringkat tetapi tidak pernah mendistribusikan peringkat apa pun. Skenario ini disebut *rank sink* (Page dkk., 1999).

2. *PageRank*

Untuk mengatasi masalah *rank sink*, diamati aktivitas pengguna. Sebuah fenomena ditemukan bahwa tidak semua pengguna mengikuti *link* yang ada. Misalnya, setelah melihat halaman a, beberapa pengguna mungkin tidak memutuskan untuk mengikuti tautan yang ada tetapi langsung menuju ke halaman b, yang tidak langsung terhubung ke halaman a. Untuk tujuan ini, pengguna cukup mengetikkan *URL* halaman b ke dalam kolom teks URL dan langsung melompat ke halaman b. Dalam hal ini, peringkat halaman b harus dipengaruhi oleh halaman a meskipun kedua halaman ini tidak terhubung langsung. Oleh karena itu, tidak ada *rank sink* yang absolut. Berdasarkan pertimbangan fenomena tersebut di atas, PageRank yang original diterbitkan (Page dkk., 1999):

$$PR(u) = (1 - d) + d \sum_{v \in B(u)} \frac{PR(v)}{N_v} \tag{2.5}$$

Di mana d merupakan *damping factor* yang biasanya bernilai 0.85. Kita juga bisa menganggap d sebagai probabilitas pengguna mengikuti tautan dan dapat menganggap $(1 - d)$ sebagai distribusi *pagerank* dari halaman yang tidak terhubung secara langsung.

Untuk menguji kegunaan algoritma *PageRank*, *Google* menerapkannya pada mesin pencari *Google*. Dalam percobaan, algoritma *PageRank* bekerja secara efisien dan efektif karena nilai peringkat konvergen ke toleransi yang wajar dalam logaritma kasar $(log\, n)$ (Page dkk., 1999). Skor peringkat halaman web dibagi secara merata di atas halaman yang ditautkannya. Meskipun algoritma *PageRank* berhasil digunakan

di *Google*, terdapat satu masalah yang masih ada: di web yang sebenarnya, beberapa tautan di halaman web mungkin lebih penting daripada yang lain (Xing dan Ghorbani, 2004).

J. Metode Scrum

Metode Scrum diciptakan oleh Jeff Sutherland dan tim pengembangnya pada awal 1990-an. Sutherland, Viktorov, Blount dan Puntikov menggambarkan Scrum sebagai "Proses pengembangan perangkat lunak *Agile* yang dirancang untuk menambah energi, fokus, kejelasan, dan transparansi bagi tim proyek yang mengembangkan sistem perangkat lunak" (Sutherland dkk., 2007). Proses Scrum mematuhi prinsip-prinsip pendekatan *Agile*. Prinsip pendekatan *Agile* berfokus pada memuaskan pelanggan melalui pengiriman awal perangkat lunak yang berharga; membolehkan perubahan kebutuhan; kolaborasi antara pengembang dan pelaku bisnis; perangkat lunak yang berfungsi (sebagai ukuran kemajuan); menjaga desain tetap sederhana; dan secara berkala, membuat tim merenungkan bagaimana menjadi lebih efektif selama proses pengembangan.

Scrum menawarkan cara yang dapat disesuaikan untuk bekerja pada proyek yang berbeda yang memiliki berbagai persyaratan dan Scrum memiliki keunggulan seperti pemilihan persyaratan atau spesifikasi kebutuhan yang fleksibel untuk *sprint* dan tidak ada prosedur khusus yang harus diikuti. Prinsipnya adalah bekerja secara iteratif hingga mencapai waktu yang ditentukan dan dapat memenuhi kebutuhan konsumen.

K. Tim Scrum

Tim Scrum setidaknya terdiri dari *Product Owner*, *Development Team* dan *Scrum Master*. Tim Scrum adalah tim yang lintas fungsi, di mana anggotanya

memiliki semua keterampilan yang diperlukan untuk melakukan pekerjaan tanpa bergantung pada orang lain di luar tim pada setiap *Sprint*. Anggota tim Scrum juga mengatur diri mereka sendiri yang berarti secara internal mereka memutuskan siapa melakukan apa, kapan, dan bagaimana. Berikut merupakan anggota-anggota dari Tim Scrum.

1. *Product Owner*

   Pemilik produk (*product owner*) adalah peran yang berfokus pada keberhasilan produk. Pemilik produk mencoba memaksimalkan nilai produk perangkat lunak bagi penggunanya. Oleh karena itu, pemilik produk bekerja secara kolaboratif dengan tim pengembangan dalam satu tim dan memberikan arahan pengembangan produk melalui visi produk mereka. Peran ini mengutamakan pekerjaan tim pengembang agar dapat diperoleh nilai terbaik dari produk perangkat lunak secara bertahap dan iteratif.

   Selain berkolaborasi dengan tim pengembangan, Pemilik produk membangun kemitraan aktif dengan para pemangku kepentingan. Pemangku kepentingan adalah siapa pun di luar tim yang tertarik pada keberhasilan produk perangkat lunak. Manajemen, pelanggan, dan sponsor produk, kita dapat menyebutnya sebagai pemangku kepentingan.

   Pemilik produk adalah individu, bukan kelompok. Pemilik produk mewakili suara para pemangku kepentingan, mencari kesamaan di antara banyak kepentingan produk melalui komunikasi yang baik.

2. *Development Team*

   Tim pengembangan (*development team*) adalah tim dengan berbagai peran yang diperlukan untuk membangun produk perangkat lunak. Beberapa peran khusus yang sering kita temui adalah *software developer*, *UI/UX designer*, dan

quality assurance. Peran-peran ini berkolaborasi setiap hari dalam tim pengembangan untuk membangun produk perangkat lunak sesuai dengan prioritas kerja yang disepakati dengan pemilik produk.

Di dalam Scrum, siapa pun yang termasuk dalam tim pengembangan disebut pengembang. Tidak ada peran khusus dalam tim pengembangan seperti yang didefinisikan oleh Scrum. Setiap orang adalah pengembang, dan siapa pun dapat melakukan apa saja untuk membangun perangkat lunak berkualitas tinggi. Membuat tim pengembangan lintas fungsi dan mengatur diri sendiri bukanlah tugas yang mudah. Ada proses yang membutuhkan waktu dan kesabaran untuk meningkatkan tim untuk mencapai level itu.

3. *Scrum Master*

   Scrum master adalah peran yang berfokus pada keberhasilan implementasi proses perangkat lunak yang dihasilkan dari metode Scrum. *Scrum master* memastikan aktivitas pengembangan dengan mengikuti proses Scrum. *Scrum master* membantu pemilik produk dan tim pengembang bekerja sama dalam kerangka proses Scrum.

   Sebagai pelatih, *scrum master* melatih tim Scrum untuk menggunakan Scrum dengan tepat dalam lingkungan organisasi sehari-hari untuk mendapatkan manfaat terbaik dari Scrum. *Scrum master* membantu semua orang di tim Scrum memahami tujuan dan manfaat Scrum untuk pengembangan produk perangkat lunak yang sukses.

   Sebagai fasilitator, *scrum master* menjadi fasilitator di tim Scrum selama pertemuan Scrum. *Scrum master* mencari keputusan antara Pemilik Produk dan Tim Pengembang untuk memberikan perangkat lunak terbaik yang dihasilkan melalui musyawarah.

Sebagai agen perubahan, *scrum master* membawa perubahan pada tim dan organisasi menuju implementasi proses Scrum yang lebih baik. Untuk organisasi baru, *scrum master* memperkenalkan organisasi ke Scrum dan membantu organisasi mengadopsi Scrum. *Scrum master* memandu proses perubahan dari proses perangkat lunak yang tidak terdefinisi ke proses Scrum.

L. Aktivitas-aktivitas Scrum (*Scrum Events*)

Aktivitas-aktivitas pada Scrum dibuat untuk menciptakan kontinuitas dan mengurangi fase lain yang tidak terdaftar di Scrum. Terjadinya kegagalan dalam menerapkan salah satu dari tahapan ini akan mengurangi transparansi dan menghilangkan peluang untuk peninjauan dan perubahan. Aktivitas-aktivitas yang terdapat pada Scrum adalah sebagai berikut.

1. *Sprint*

   Sprint merupakan tahap pengembangan perangkat lunak dengan durasi maksimum satu bulan dan memiliki durasi yang konsisten selama proses pengembangan produk. *Sprint* baru akan langsung dimulai segera setelah *Sprint* sebelumnya selesai. Sprint terdiri dari *Sprint Planning*, *Daily Scrum*, *Sprint Review* dan *Sprint Retrospective*.

2. *Sprint Planning*

   Pekerjaan yang dilakukan dalam *Sprint* direncanakan dalam tahap *Sprint Planning*. Rencana tersebut dibuat oleh semua anggota dalam tim Scrum. Untuk *Sprint* yang berdurasi satu bulan, batas waktu *Sprint Planning* maksimal 8 jam.

3. *Daily Scrum*

Daily Scrum adalah aktivitas yang berlangsung hingga 15 menit dan bertujuan agar tim pengembang dapat menyinkronkan pekerjaan mereka dan membuat rencana untuk 24 jam ke depan. Hal ini dilakukan dengan meninjau pekerjaan sejak tahap *Daily Scrum* terakhir dan memperkirakan apa yang dapat dilakukan sebelum melanjutkan ke *Daily Scrum* berikutnya.

4. *Sprint Review*

   Sprint Review diadakan di akhir setiap *Sprint* untuk meninjau iterasi dan merevisi *Product Backlog* sesuai kebutuhan. Selama *Sprint Review*, Tim Scrum dan pemangku kepentingan berkolaborasi untuk membahas pekerjaan yang telah dilakukan dalam *Sprint* yang baru saja selesai. Berdasarkan hasil ini dan setiap perubahan pada *Product Backlog* selama Sprint, peserta berkolaborasi untuk menentukan apa yang dapat dilakukan di *Sprint* berikutnya untuk mengoptimalkan nilai produk. Pertemuan bersifat informal, bukan pertemuan status, dan pemaparan diharapkan dapat menghimpun masukan dan menumbuhkan semangat kerja sama.

5. *Sprint Retrospective*

   Sprint Retrospective adalah kesempatan bagi Tim Scrum untuk meninjau diri mereka masing-masing dan mengembangkan rencana untuk perbaikan di *Sprint* berikutnya. *Sprint Retrospective* dilakukan setelah *Sprint Review* selesai dan sebelum *Sprint Planning* berikutnya. Batas waktu maksimum tahap ini adalah 3 jam untuk *Sprint* yang berdurasi satu bulan. Untuk *Sprint* yang lebih pendek, batas waktunya biasanya lebih pendek. *Scrum Master* memastikan bahwa langkah-langkah ini dilakukan dan semua anggota memahami tujuan mereka. *Scrum Master* mendidik Tim Scrum untuk mengimplementasikannya dalam batas waktu yang telah ditentukan. *Scrum*

Master berpartisipasi sebagai mitra yang bertanggung jawab atas setiap proses yang terjadi saat Scrum terjadi.

M. Komponen Scrum

Komponen Scrum mewakili pekerjaan atau nilai. Mereka dirancang untuk memaksimalkan transparansi informasi penting. Jadi setiap orang yang mengkajinya memiliki dasar adaptasi yang sama. Berikut merupakan komponen-komponen yang terdapat pada metode Scrum.

1. *Product Backlog*

   Product Backlog adalah daftar apa yang dibutuhkan untuk meningkatkan produk. *Item* dari *Product Backlog* yang dapat dijalankan oleh Tim Scrum dalam sebuah *Sprint* dianggap siap untuk diseleksi dalam aktivitas *Sprint Planning*. Mereka biasanya mendapatkan transparansi ini setelah melakukan kegiatan revisi. Perbaikan *Product Backlog* adalah operasi yang memecah dan mendefinisikan lebih lanjut *item-item* Product Backlog menjadi *item* yang lebih kecil dan lebih tepat. Menambahkan detail seperti deskripsi, urutan, dan ukuran diperlukan. Pengembang yang akan melakukan pekerjaan bertanggung jawab atas pengukuran. Pemilik produk dapat memengaruhi pengembang dengan membantu mereka memahami dan memilih.

2. *Sprint Backlog*

   Sprint Backlog mencakup tujuan Sprint (mengapa), item *Product Backlog* yang dipilih untuk *Sprint* (apa), dan rencana yang layak untuk memberikan *Increment* (bagaimana). *Sprint Backlog* adalah rencana yang dibuat oleh pengembang. Ini adalah gambaran waktu nyata dari pekerjaan yang direncanakan pengembang selama *Sprint* untuk mencapai tujuan *Sprint*. Oleh

karena itu, *Sprint Backlog* diperbarui sepanjang *Sprint* karena semakin banyak yang dipelajari. Ini harus cukup detail sehingga mereka dapat memeriksa seluruh kemajuan mereka di *Daily Scrum*.

3. *Increment*

   Peningkatan (*Increment*) adalah komponen yang dianggap sebagai batu loncatan untuk mencapai tujuan produk. Setiap peningkatan melengkapi semua tambahan sebelumnya dan diverifikasi secara menyeluruh untuk memastikan semua tambahan berfungsi secara maksimal. Beberapa penambahan dapat dilakukan di *Sprint*. Total yang ditambahkan diberikan dalam diskusi *Sprint*. Namun, Peningkatan dapat dikirim ke pemangku kepentingan sebelum *Sprint* selesai. Diskusi *Sprint* tidak boleh dianggap sebagai referensi untuk evaluasi. (Schwaber dan Sutherland, 2020)

BAB III

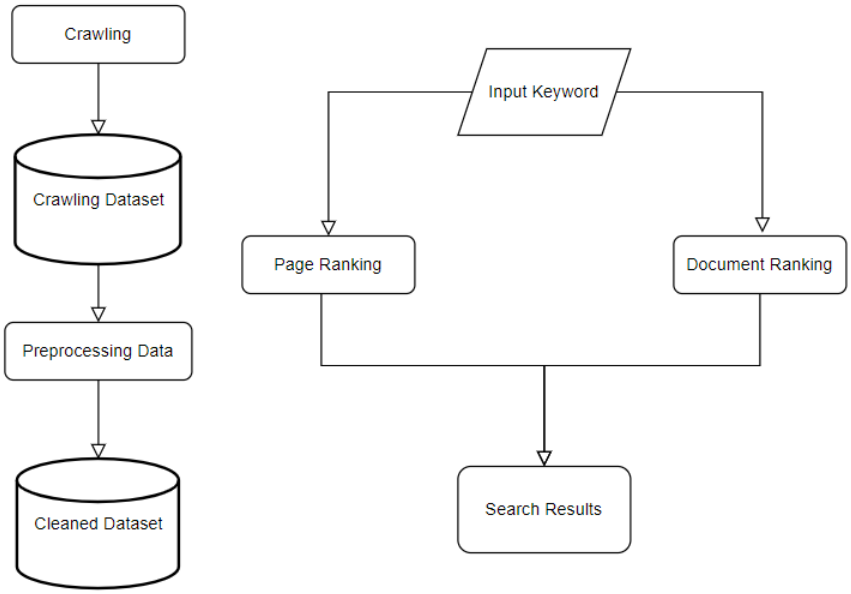
METODOLOGI PENELITIAN

A. Deskripsi Sistem

Gambar 3.1: *Wireframe* Tampilan Sistem

Sistem yang akan dibuat pada penelitian ini adalah sistem *search engine* yang berfungsi untuk pencarian *website* berbasis *terminal* yang akan menerima *input* berupa *keyword* dari pengguna dan sistem akan menampilkan hasil *website* yang relevan sesuai dengan *keyword* tersebut.

Pembuatan *search engine* ini menggunakan beberapa algoritma dan metode sebagai pendukungnya, yaitu algoritma *modified similarity based crawling* yang terdapat pada komponen *crawler*, metode *TF-IDF* untuk memberikan bobot pada dokumen berdasarkan *keyword*, dan metode *Google PageRank* untuk perangkingan *website*. Selain itu, proses pengembangan sistem ini memakai metode pengembangan Scrum.

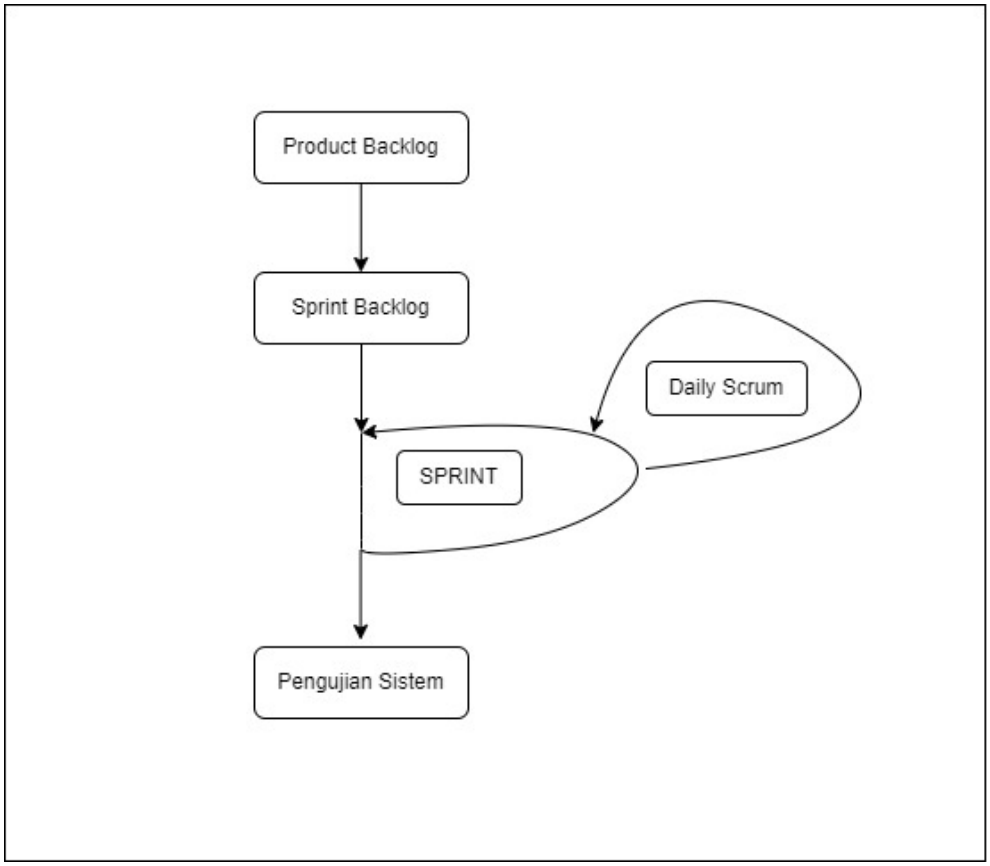


Gambar 3.2: *Flowchart* Sistem

Proses dari sistem *search engine* dimulai dengan mengumpulkan data ke dalam *database* sebanyak-banyaknya menggunakan *crawler*, lalu data tersebut akan diolah melalui *preprocessing data* dan disimpan ke dalam *database*. Setelah data sudah siap, maka sistem dimulai dengan meminta *input* berupa *keyword* dari pengguna lalu *keyword* tersebut akan diproses melalui *page ranking* dan *document ranking*. Setelah itu hasil pencarian akan muncul sesuai dengan *keyword* yang dimasukkan.

B. Desain Penelitian

Agar penelitian lebih terstruktur dan memudahkan penulis melakukan penelitian, maka dibutuhkan desain penelitian. Desain penelitian mencakup tahapan-tahapan yang dilakukan penulis dalam pembuatan sistem dengan menggunakan metode Scrum. Tahapan tersebut dapat dilihat pada gambar 3.3.



Gambar 3.3: Desain Penelitian

Tahap pertama yang dilakukan pada penelitian ini adalah mendefinisikan *product backlog* dari sistem yang akan dibuat, lalu membuat jadwal pengerjaan atau *sprint backlog*. Setelah itu, *sprint* dimulai dengan diselingi *daily scrum* untuk melaporkan *progress* dan menentukan *task* selanjutnya. Setelah sistem selesai dikerjakan di dalam *sprint*, maka dilakukan pengujian sistem.

C. Alat dan Bahan Penelitian

Dalam penelitian ini, beberapa alat berupa perangkat keras yang digunakan untuk menunjang pembuatan sistem adalah sebagai berikut:

1. Laptop yang mempunyai CPU Intel i7-6700hq dan RAM 20GB

2. LCD Monitor dengan resolusi 1920 x 1080 *pixel*

3. Koneksi berbasis *Wi-Fi*

Perangkat keras di atas sudah memenuhi persyaratan untuk menjalankan aplikasi *Python* sebagai *server* maupun sebagai *interpreter* dan dapat menjalankan *server* untuk *database MySQL*. Selain perangkat keras, perangkat lunak yang dipakai untuk membuat sistem ini adalah sebagai berikut:

1. Windows 10 64-bit Operating System

2. Visual Studio Code sebagai *code editor*

3. XAMPP untuk menjalankan *MySQL database*

4. Python 3 untuk menjalankan program Python

D. Perancangan Sistem

Perancangan sistem pada penelitian ini sesuai dengan komponen yang ada di dalam metode Scrum. Komponen-komponen tersebut terdiri dari *product backlog*, *sprint backlog*, *sprint*, *daily scrum*, dan *pengujian sistem*. Berikut merupakan rincian dari perancangan sistem sesuai dengan komponen Scrum.

1. *Product Backlog*

Berikut merupakan tabel *product backlog* sistem *search engine* yang telah dibuat berdasarkan diskusi dengan *Scrum Master*. Transkrip percakapan dengan *Scrum Master* terdapat pada **Lampiran A**. Setiap *item* dari *product backlog* ini akan diimplementasikan pada *sprint* dari awal hingga selesai.

Tabel 3.1: *Product Backlog*

No	***Story***	***Sprint***	***Status***	***Priority***
1	Mengubah struktur project dari program *crawler* menjadi modular	1	Completed	Penting
2	Konfigurasi program *crawler* sebagai *background service* di sistem operasi	2		Penting
3	Perancangan struktur *package*, *class*, dan fungsi program *TF IDF*	3		Penting
4	Perancangan struktur *package*, *class*, dan fungsi program *Page Rank*	4		Penting
5	Konfigurasi program *page rank* sebagai *background service* di sistem operasi	5		Penting
6	Merancang *REST API* untuk fungsi utama *TF IDF*	6		Penting
7	Perancangan struktur *project* arsitektur *search engine*	7		Penting
8	Perancangan struktur kode untuk modul pengindeks *Generalized Suffix Tree*	8		Penting

Berdasarkan tabel 3.1 di atas, *Product Backlog* pada penelitian ini terdiri dari 4 komponen, yaitu *story*, *sprint*, *status*, dan *priority*. *Story* menggambarkan pekerjaan besar yang nantinya akan dipecah lagi menjadi *task-task* yang kecil di dalam *Sprint*. Komponen *sprint* pada tabel ini menandakan *sprint* berapa *story* tersebut akan dilaksanakan. Komponen *status* menjelaskan apakah *story* tersebut sudah selesai atau belum. Sedangkan komponen *priority* menjelaskan skala prioritas dari *story* tersebut.

2. *Sprint Backlog*

Sprint backlog adalah daftar pekerjaan yang perlu dikerjakan ataupun yang sudah dikerjakan pada *sprint*. Di dalam *sprint backlog*, berbagai *task* kecil dibuat berdasarkan *story* yang terdapat pada *product backlog*. Berikut merupakan rincian dari *sprint backlog* yang pertama.

1. *Sprint-1*

   Sprint-1 dilakukan sepekan pada tanggal 16 April 2022 sampai dengan 22 April 2022. *Story* pertama pada *product backlog* dipecah menjadi beberapa *task* sebagai berikut.

Tabel 3.2: *Sprint 1 Backlog*

No	***Story***	***Task***	***Status***
1	Mengubah struktur kode dari program *crawler* menjadi modular	Membuat rancangan class diagram	Completed
2		Membuat rancangan entity relationship diagram untuk database	Completed
3		Mengubah struktur kode crawler sesuai dengan rancangan class diagram	Completed
4		Implementasi threading untuk menjalankan lebih dari satu proses dalam satu waktu	Completed
5		Testing program versi terbaru	Completed

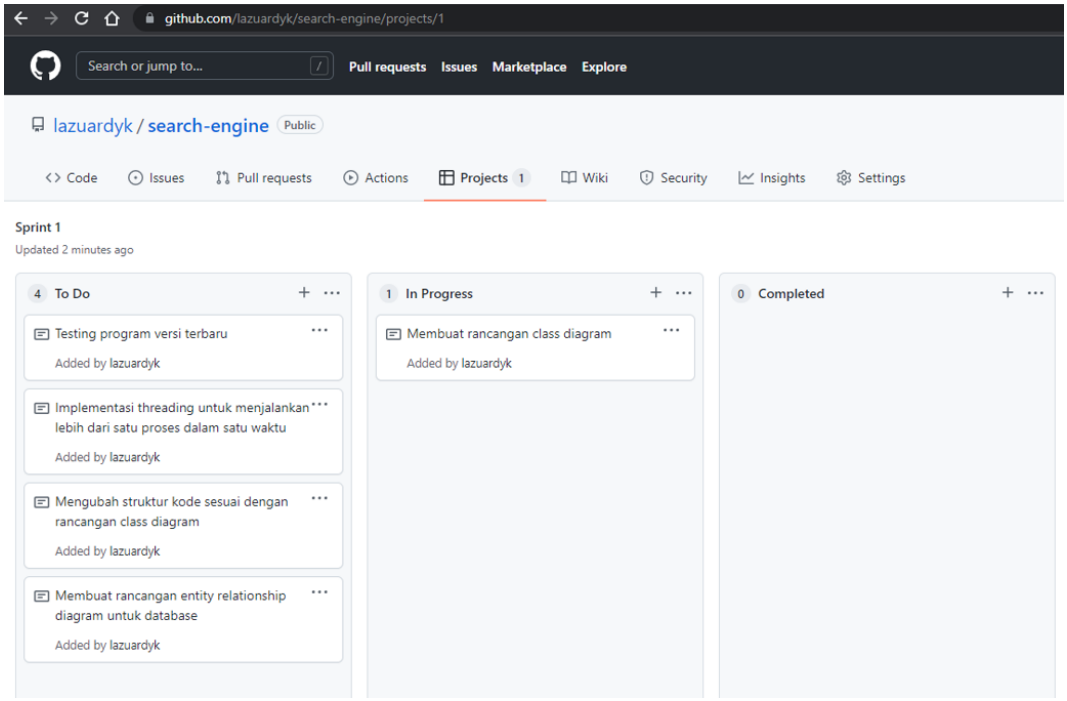
3. *Daily Scrum*, *Sprint Review*, dan *Sprint Retrospective*

Dikarenakan sumber daya dan waktu yang terbatas, aktivitas *daily scrum* akan digantikan dengan cara mencatat hal-hal penting dan hambatan yang terjadi setiap hari sebagai *note* di *Github Projects*.

Sprint review, dan *sprint retrospective* dilakukan pada awal pekan yaitu di hari Selasa melalui *voice call*. Pada awal pekan ini akan dilakukan evaluasi mengenai perkembangan sistem maupun hambatan yang terjadi selama pengerjaan di setiap *sprint*.

4. *Sprint 1 Report*

Setelah merencanakan *sprint backlog*, pekerjaan pada *sprint* ini harus mengikuti rencana kerja yang ditentukan oleh *sprint backlog*. Tujuan dari *sprint-1* ini adalah mengubah struktur kode program *crawler* sebelumnya menjadi modular. Untuk manajemen *task* dan *progress*, penulis menggunakan *GitHub Projects* seperti pada gambar 3.4.



Gambar 3.4: *Github Projects Sprint-1*

Terdapat 3 kolom pada *project* yang dibuat, yaitu *to do*, *in progress*, dan *completed*. Setiap *task* yang perlu dikerjakan akan ditulis dan dimasukkan ke dalam kolom *to do*, *task* yang sedang dikerjakan akan dipindahkan ke kolom *in progress*, dan jika task sudah selesai akan dipindahkan ke kolom *completed*. Berikut merupakan hasil dari pengerjaan yang dilakukan selama *sprint 1*.

1. *Entity Relationship Diagram Database*

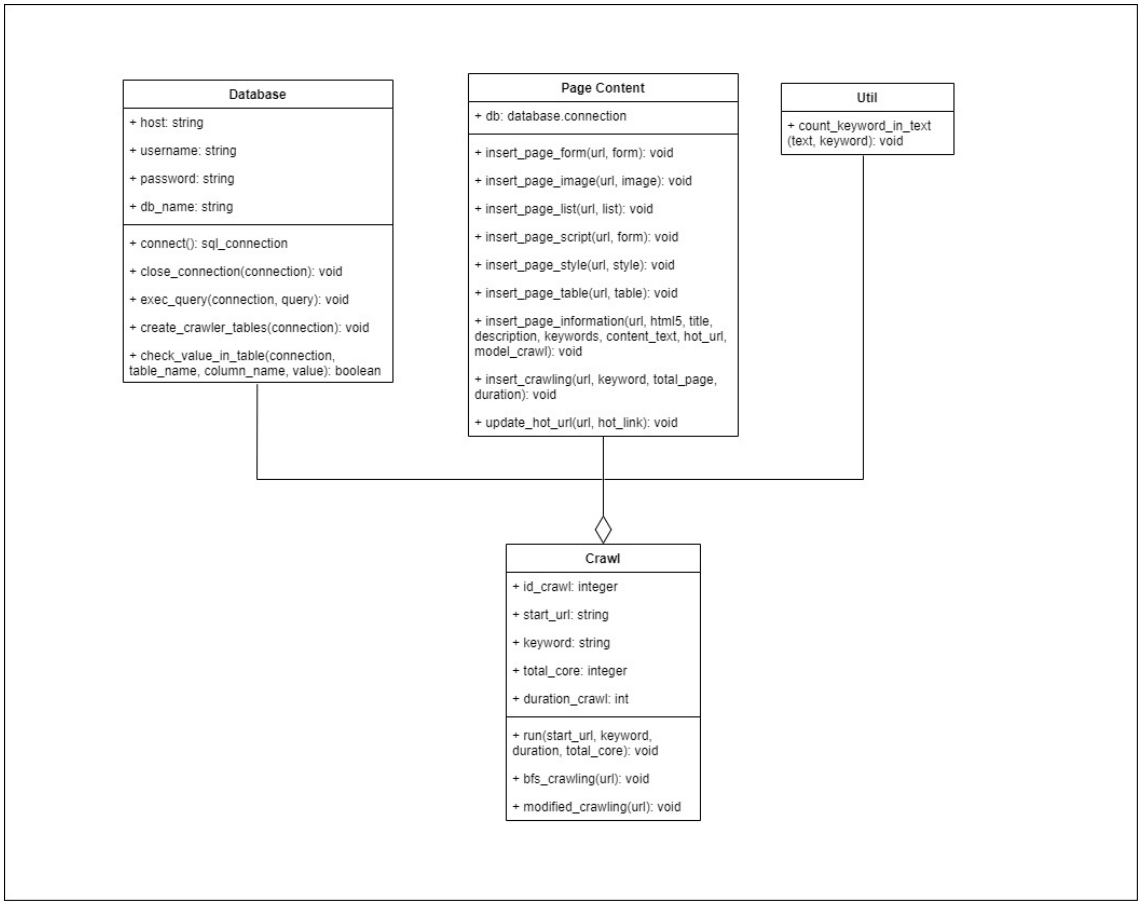
   Entity Relationship Diagram (ERD) pada *sprint-1* ini menggambarkan masing-masing entitas dan relasi antar entitas pada *database* untuk *crawler*.

Gambar 3.5: *Entity Relationship Diagram Sprint-1*

Tabel-tabel yang dibutuhkan untuk menyimpan data pada *crawler* yaitu tabel *crawling*, *page_linking*, *page_information*, *page_images*, *page_tables*, *page_styles*, *page_forms*, *page_list*, dan *page_scripts*.

2. *Class Diagram*

Class diagram pada gambar 3.6 menggambarkan kelas-kelas dalam sistem yang dipakai *crawler*. Rancangan tersebut merupakan rancangan *class diagram* dari program *crawler* yang modular.



Gambar 3.6: *Class Diagram Sprint-1*

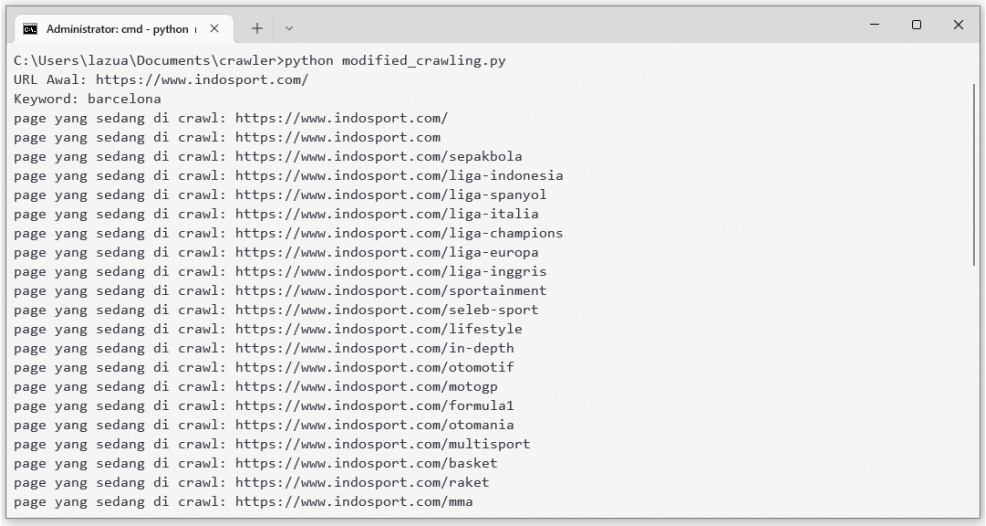
Pada rancangan *class diagram crawler* ini, terdapat 4 kelas yang akan dibuat, yaitu kelas *database* untuk koneksi dan pembuatan *table* pada *database*, kelas *page content* yang berisi fungsi-fungsi untuk menyimpan konten dari halaman *website*, kelas *util* yang terdapat fungsi-fungsi pendukung, dan kelas *crawl* sebagai kelas utama untuk menjalankan *crawler*.

3. *Source code*

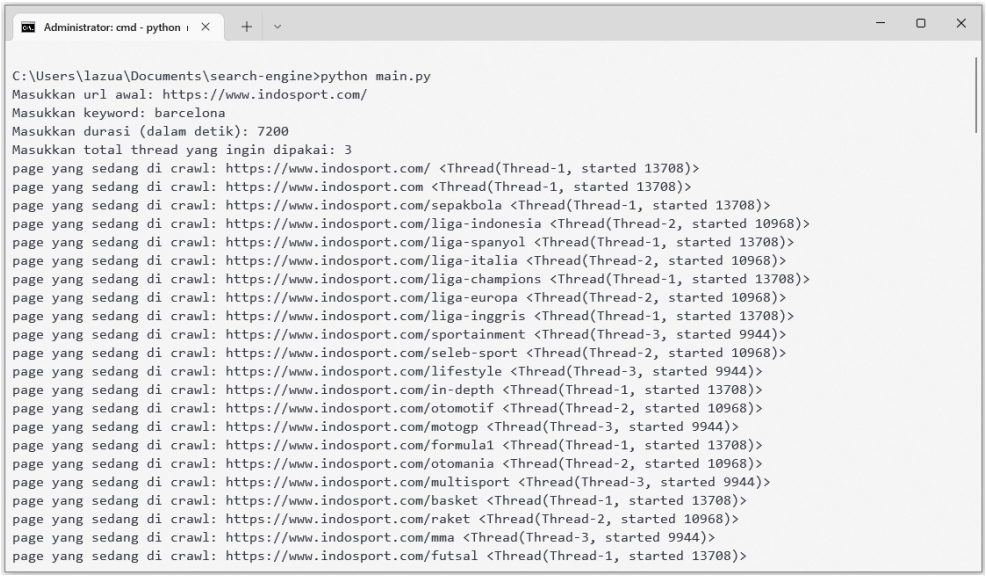
   Source code dari *sprint-1* terdapat di *Github Repository*, yang dapat diakses pada link https://github.com/lazuardyk/search-engine/tree/sprint-1

4. Pengujian Sistem *Crawler*

Pengujian sistem dilakukan dengan menjalankan program *crawler* yang dilakukan penelitian sebelumnya dan versi yang terbaru melalui *terminal command prompt* dan menargetkan situs https://www.indosport.com/ dengan *keyword* berupa "barcelona". Kedua program tersebut dijalankan selama 2 jam dan pada versi crawler terbaru menggunakan 3 *threads* karena menerapkan konsep *multithreading*.

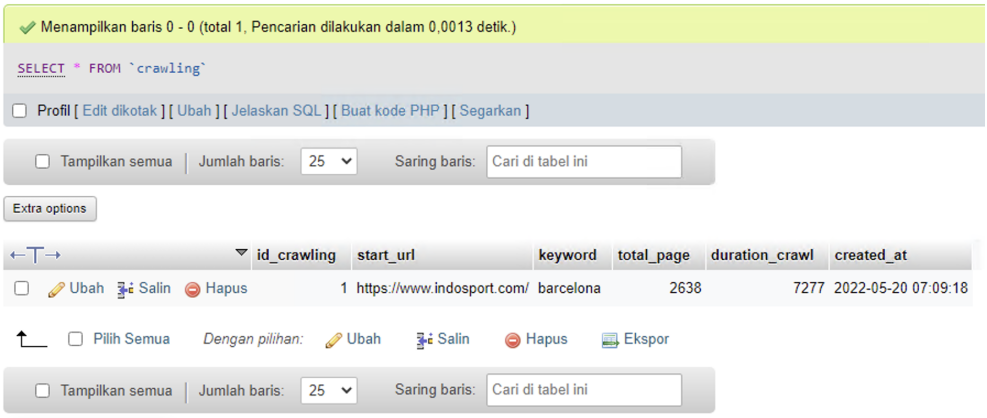


Gambar 3.7: Pengujian *crawler* penelitian sebelumnya

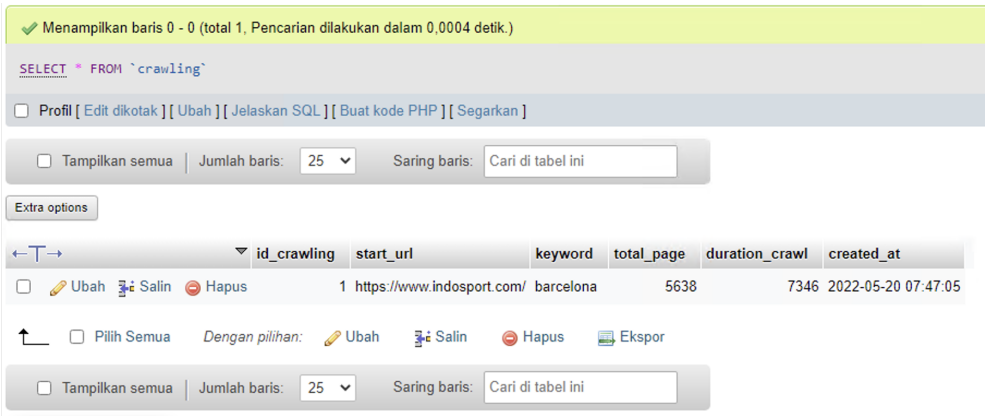


Gambar 3.8: Pengujian *crawler* versi terbaru

Gambar 3.7 dan gambar 3.8 merupakan tampilan *crawler* versi lama dan terbaru yang berjalan setelah dimasukkan *input* dari *user*. Hasil dari kedua *crawler* yang telah selesai dijalankan ini terdapat pada gambar 3.9 dan gambar 3.10.



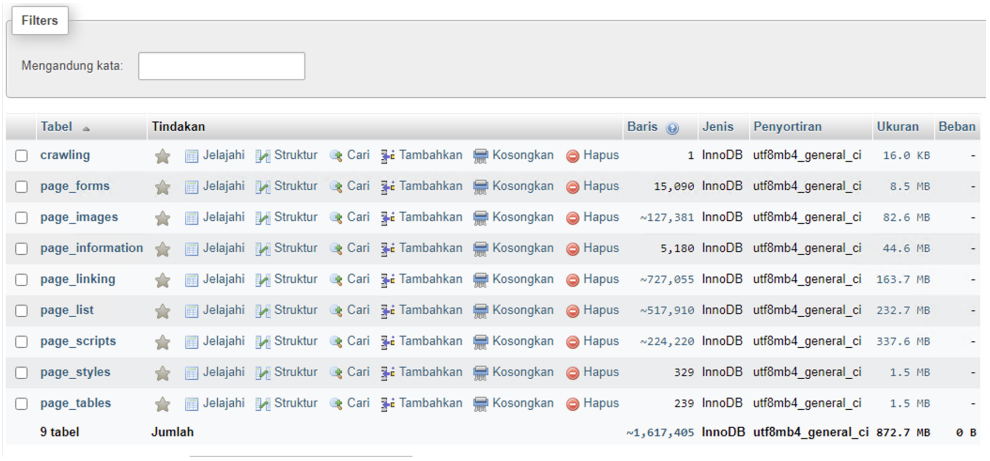
Gambar 3.9: Hasil pengujian *crawler* penelitian sebelumnya



Gambar 3.10: Hasil pengujian *crawler* versi terbaru

Untuk situs awal https://www.indosport.com/ hasil yang didapat pada *crawler* penelitian sebelumnya adalah 2638 halaman dengan durasi 7277 detik, sedangkan pada *crawler* versi terbaru adalah 5638 halaman dengan durasi 7346 detik. Dengan demikian, waktu yang dibutuhkan untuk *crawling* satu halaman adalah 2,7 detik pada *crawler* versi lama dan 1,3 detik pada *crawler*

terbaru yang saat ini dikembangkan.


Filters

Mengandung kata:

	Tabel	Tindakan	Baris	Jenis	Penyortiran	Ukuran	Beban
☐	crawling	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	1	InnoDB	utf8mb4_general_ci	16.0 KB	-
☐	page_forms	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	15,090	InnoDB	utf8mb4_general_ci	8.5 MB	-
☐	page_images	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	~127,381	InnoDB	utf8mb4_general_ci	82.6 MB	-
☐	page_information	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	5,180	InnoDB	utf8mb4_general_ci	44.6 MB	-
☐	page_linking	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	~727,055	InnoDB	utf8mb4_general_ci	163.7 MB	-
☐	page_list	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	~517,910	InnoDB	utf8mb4_general_ci	232.7 MB	-
☐	page_scripts	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	~224,220	InnoDB	utf8mb4_general_ci	337.6 MB	-
☐	page_styles	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	329	InnoDB	utf8mb4_general_ci	1.5 MB	-
☐	page_tables	Jelajahi Struktur Cari Tambahkan Kosongkan Hapus	239	InnoDB	utf8mb4_general_ci	1.5 MB	-
	9 tabel	Jumlah	~1,617,405	InnoDB	utf8mb4_general_ci	872.7 MB	0 B

Gambar 3.11: Hasil pengujian *crawler* terbaru 2

Konten *website* yang diperoleh dari proses pengujian *crawling* pada *crawler* versi terbaru untuk setiap tabelnya adalah 15.090 baris pada *page_forms*, 127.381 baris pada *page_images*, 5.180 baris pada *page_information*, 727.055 baris pada *page_linking*, 517.910 baris pada *page_list*, 224.220 baris pada page_scripts, 329 baris pada *page_styles*, dan 239 baris pada *page_tables*.

LAMPIRAN A

Transkrip Percakapan

Hari: Sabtu

Tanggal: 16 April 2022

PL: Penulis

SM: Scrum Master

PL: Bagaimana cara kerja sistem search engine ini?

SM: Kita akan membuat search engine yang berjalan di terminal dan menerima keyword pengguna dan juga bisa diakses melalui API sehingga bisa berguna jika ingin diintegrasikan ke tampilan yang lain.

PL: Apakah sistem ini akan berjalan di lokal atau online?

SM: Untuk saat ini, sistem akan berjalan di lokal.

PL: Apa saja yang dibutuhkan di product backlog?

SM: Story yang pertama adalah mengubah struktur kode program crawler yang sudah ada menjadi modular, yang kedua mengintegrasikan document ranking TF IDF ke dalam arsitektur, yang ketiga mengintegrasikan PageRank ke dalam arsitektur.

PL: Bagaimana prioritas masing-masing story tersebut?

SM: Sama penting.

PL: Apakah ada story lainnya?

SM: Story keempat adalah konfigurasi program sebagai background service di sistem operasi, dan ke yang lima untuk merancang REST API untuk fungsi utamanya.

PL: Apa saja fungsi utama dari API nya?

SM: Fungsi utamanya adalah melihat peringkat website berdasarkan PageRank, mencari dokumen yang relevan, dan melihat peringkat website secara overall.

PL: Baik. Bahasa pemrograman apa yang akan dipakai pada sistem ini?

SM: Karena web crawler dan komponen yang lain memakai Python, maka kita akan memakai bahasa Python 3 juga.

DAFTAR PUSTAKA

Brin, S. dan Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Castillo, C. (2005). Effective web crawling. In *Acm sigir forum*, volume 39, pages 55–56. Acm New York, NY, USA.

Cho, J., Garcia-Molina, H., dan Page, L. (1998). Efficient crawling through url ordering.

Eichmann, D. (1994). The rbse spider - balancing effective search against web load. *Proceedings of the first World Wide Web Conference*.

Feldman, R. dan Sanger, J. (2007). The text mining handbook- advanced approaches in analyzing unstructured data.

Kaur, S. dan Geetha, G. (2020). Simhar-smart distributed web crawler for the hidden web using sim+ hash and redis server. *IEEE Access*, 8:117582–117592.

Manning, C., Raghavan, P., dan Schutze, H. (2009). An introduction to information retrieval (online edition). *Cambridge: Cambridge University Press*.

Page, L., Brin, S., Motwani, R., dan Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Pinkerton, B. (1994). Finding what people want: Experiences with the webcrawler. In *Proc. 2nd WWW Conf., 1994*.

Qaiser, S. dan Ali, R. (2018). Text mining: Use of tf-idf to examine the relevance of words to documents. *International Journal of Computer Applications (pp. 25-29)*.

Qoriiba, Muhammad, F. (2021). Perancangan crawler sebagai pendukung pada search engine.

Schwaber, K. dan Sutherland, J. (2020). *The Definitive Guide to Scrum: The Rules of the Game*. ScrumGuides.org.

Seymour, T., Frantsvog, D., dan Kumar, S. (2011). History of search engines. *International Journal of Management & Information Systems (IJMIS)*, 15(4):47–58.

Statista (2022). Worldwide market share of search engine.

Sutherland, J., Viktorov, A., Blount, J., dan Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, pages 274a–274a.

Xing, W. dan Ghorbani, A. (2004). Weighted pagerank algorithm. In *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, pages 305–314.