

**PERANCANGAN ARSITEKTUR SEARCH ENGINE DENGAN
MENGINTEGRASIKAN WEB CRAWLER, ALGORITMA PAGE
RANKING, DAN DOCUMENT RANKING**

Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



**Oleh:
Lazuardy Khatulistiwa
1313618008**

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

2023

LEMBAR PERSETUJUAN HASIL SIDANG SKRIPSI

PERANCANGAN ARSITEKTUR *SEARCH ENGINE* DENGAN MENGINTEGRASIKAN *WEB CRAWLER*, ALGORITMA *PAGE RANKING*, DAN *DOCUMENT RANKING*

Nama : Lazuardy Khatulistiwa
No. Registrasi : 1313618008

	Nama	Tanda Tangan	Tanggal
Penanggung Jawab			
Dekan	: <u>Prof. Dr. Muktiningsih N, M.Si.,</u> NIP. 196405111989032001
Wakil Penanggung Jawab			
Wakil Dekan I	: <u>Dr. Esmar Budi, S.Si., MT.</u> NIP. 197207281999031002
Ketua	: <u>Ir. Fariani Hermin Indiyah, M.T.</u> NIP. 196002111987032001		16 / 02 / 2023
Sekretaris	: <u>Ari Hendarno, S.Pd., M.Kom.</u> NIP. 198811022022031002		16 / 02 / 2023
Penguji	: <u>Med Irzal, M.Kom.</u> NIP. 197706152003121001		17 / 02 / 2023
Pembimbing I	: <u>Muhammad Eka Suryana, M.Kom.</u> NIP. 198512232012121002		20 / 02 / 2023
Pembimbing II	: <u>Drs. Mulyono, M.Kom.</u> NIP. 196605171994031003		20 / 02 / 2023

Dinyatakan lulus ujian skripsi tanggal: 31 Januari 2023

LEMBAR PERNYATAAN

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul "**Perancangan Arsitektur Search Engine dengan Mengintegrasikan Web Crawler, Algoritma Page Ranking, dan Document Ranking**" yang disusun sebagai syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Ilmu Komputer Universitas Negeri Jakarta adalah karya ilmiah saya dengan arahan dari dosen pembimbing.

Sumber informasi yang diperoleh dari penulis lain yang telah dipublikasikan dan disebutkan dalam teks skripsi ini, telah dicantumkan dalam Daftar Pustaka sesuai dengan norma, kaidah dan etika penulisan ilmiah.

Jika dikemudian hari ditemukan sebagian besar skripsi ini bukan hasil karya saya sendiri dalam bagian-bagian tertentu, saya bersedia menerima sanksi pencabutan gelar akademik yang saya sanding dan sanksi-sanksi lainnya sesuai dengan peraturan perundangan yang berlaku.

Jakarta, 19 Januari 2023

Lazuardy Khatulistiwa

1313618008

HALAMAN PERSEMPAHAN

Untuk Keluargaku dan Diriku Sendiri.

KATA PENGANTAR

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

Keberhasilan dalam menyusun skripsi ini tidak lepas dari bantuan berbagai pihak yang mana dengan tulus dan ikhlas memberikan masukan guna sempurnanya 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 skripsi ini.
4. Yth. Bapak Drs. Mulyono, M.Kom selaku Dosen Pembimbing II yang telah membimbing, mengarahkan, serta memberikan saran dan koreksi terhadap skripsi ini.
5. Ayah dan Ibu penulis yang selama ini telah mendukung dan membantu menyelesaikan skripsi ini.
6. Teman-teman Program Studi Ilmu Komputer 2018 yang telah mendukung dan membantu skripsi ini.

Penulis menyadari bahwa penyusunan 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 skripsi ini.

Jakarta, 19 Januari 2023

Lazuardy Khatulistiwa

ABSTRAK

LAZUARDY KHATULISTIWA. Perancangan Arsitektur *Search Engine* dengan Mengintegrasikan *Web Crawler*, Algoritma *Page Ranking*, dan *Document Ranking*. Skripsi. Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. 2023. Di bawah bimbingan Muhammad Eka Suryana, M.Kom dan Drs. Mulyono, M.Kom.

Search engine diciptakan sejak 1990 dan terus berkembang hingga hari ini. *Google* adalah search engine yang dibangun pada tahun 1998 dan merupakan search engine yang paling populer saat ini. Fungsi utama dari search engine adalah sebagai penyedia informasi berdasarkan kata kunci tertentu. Penelitian ini bertujuan untuk membuat arsitektur *search engine* dengan menggunakan *web crawler*, *page ranking*, dan *document ranking*. Algoritma yang digunakan pada *web crawler* adalah *breadth first search* dan *modified similarity based crawling*, pada *page ranking* adalah *Google PageRank*, dan pada *document ranking* adalah *TF IDF*. Proses pengembangan sistem ini menggunakan metode *Scrum* dan seluruh program yang dibuat menggunakan bahasa pemrograman *Python*. Hasil akhir dari arsitektur ini mencakup struktur *project*, rancangan diagram, konfigurasi *server*, *web service* berupa *REST API*, program *console* dan tampilan web sederhana, serta dokumentasi lengkap *project*.

Kata kunci: *search engine*, *information retrieval*, *breadth first search*, *modified similarity based crawling*, *page rank*, *TF IDF*, *scrum*

ABSTRACT

LAZUARDY KHATULISTIWA. *Search Engine Architecture Design by Integrating Web Crawler, Page Ranking Algorithm, and Document Ranking. Thesis. Faculty of Mathematics and Natural Sciences, State University of Jakarta. 2023. Supervised by Muhammad Eka Suryana, M.Kom and Drs. Mulyono, M.Kom.*

The search engine was created in 1990 and continues to grow to this day. Google is a search engine that was built in 1998 and is a search engine the most popular at the moment. The main function of the search engine is as provider of information based on certain keywords. This research aims to create a search engine architecture using web crawlers, page ranking, and document rankings. The algorithm used in the web crawler is breadth first search and modified similarity based crawling, on page ranking is Google PageRank, and on document ranking is TF IDF. The process of developing this system uses the method Scrum and all programs created using the Python programming language. The end result of this architecture includes project structure, diagram design, server configuration, web service in the form of REST API, console program and simple front-end website, as well as complete documentation of the project.

Keywords: *search engine, information retrieval, breadth first search, page rank, TF IDF, scrum*

DAFTAR ISI

LEMBAR PERNYATAAN	iii
HALAMAN PERSEMBAHAN	iv
KATA PENGANTAR	vi
ABSTRAK	vii
ABSTRACT	viii
DAFTAR ISI	x
DAFTAR GAMBAR	xiii
DAFTAR TABEL	xiv
I PENDAHULUAN	1
A. Latar Belakang Masalah	1
B. Rumusan Masalah	6
C. Pembatasan Masalah	6
D. Tujuan Penelitian	7
E. Manfaat Penelitian	7
II KAJIAN PUSTAKA	8
A. Pengertian <i>Search Engine</i>	8
B. Sejarah <i>Search Engine</i>	8
C. Arsitektur <i>Search Engine</i>	12
D. <i>Web Crawler</i>	14

E. Arsitektur <i>Web Crawler</i>	15
F. Algoritma <i>Crawling</i>	16
G. <i>Preprocessing Data</i>	19
H. Pembobotan <i>Term Frequency-Inverse Document Frequency</i>	21
I. <i>Google PageRank</i>	23
1. <i>Simplified PageRank</i>	25
2. <i>PageRank</i>	25
J. Metode Scrum	26
K. Tim Scrum	27
L. Aktivitas-aktivitas Scrum (<i>Scrum Events</i>)	30
M. Komponen Scrum	32
N. <i>Cosine Similarity</i>	33
III METODOLOGI PENELITIAN	35
A. Deskripsi Sistem	35
B. Desain Penelitian	37
C. Alat dan Bahan Penelitian	38
D. Perancangan Sistem	38
1. <i>Product Backlog</i>	39
2. <i>Sprint Backlog</i>	40
3. <i>Daily Scrum, Sprint Review, dan Sprint Retrospective</i>	41
4. <i>Sprint 1 Report</i>	42
E. Pengujian Sistem	48
IV Hasil dan Pembahasan	50
A. Perancangan Sistem	50
1. <i>Sprint reports</i>	50

B. Pembahasan Arsitektur	73
1. Struktur Direktori dan <i>File</i>	73
2. <i>Overall Similarity Score</i>	74
3. <i>Technology Stack</i>	78
4. Cara Kerja	79
C. Pengujian Arsitektur	79
1. <i>Functional Testing</i>	79
2. <i>Non-functional Testing</i>	80
D. Hasil Keseluruhan	85
V KESIMPULAN DAN SARAN	97
A. Kesimpulan	97
B. Saran	97
LAMPIRAN	100
A Transkrip Percakapan	100
B Dokumentasi <i>Testing</i>	102
C Kode untuk Uji Keandalan Server	103

DAFTAR GAMBAR

Gambar 1.1	<i>High Level Google Architecture</i> (Brin dan Page, 1998)	5
Gambar 2.1	<i>Market share search engine</i> tahun 2010 hingga 2022 (Statista, 2022)	11
Gambar 2.2	<i>High Level Google Architecture</i> (Brin dan Page, 1998)	12
Gambar 2.3	<i>High Level Architecture of Web Crawler</i> (Castillo, 2005)	15
Gambar 2.4	Tahapan <i>preprocessing data</i>	20
Gambar 2.5	Contoh <i>backlink</i>	24
Gambar 3.1	<i>Wireframe</i> Tampilan Sistem	35
Gambar 3.2	<i>Flowchart</i> Sistem	36
Gambar 3.3	Desain Penelitian	37
Gambar 3.4	<i>Github Projects Sprint-1</i>	42
Gambar 3.5	<i>Entity Relationship Diagram Sprint-1</i>	43
Gambar 3.6	<i>Class Diagram Sprint-1</i>	44
Gambar 3.7	Pengujian <i>crawler</i> penelitian sebelumnya	45
Gambar 3.8	Pengujian <i>crawler</i> versi terbaru	46
Gambar 3.9	Hasil pengujian <i>crawler</i> penelitian sebelumnya	46
Gambar 3.10	Hasil pengujian <i>crawler</i> versi terbaru	47
Gambar 3.11	Hasil pengujian <i>crawler</i> terbaru 2	47
Gambar 4.1	Kode fitur <i>start</i> dan <i>resume crawler</i> 1	51
Gambar 4.2	Kode fitur <i>start</i> dan <i>resume crawler</i> 2	52
Gambar 4.3	Kode fitur <i>stop crawler</i>	52
Gambar 4.4	Struktur <i>file crawler</i>	53
Gambar 4.5	<i>File crawler</i> di <i>server</i>	53

Gambar 4.6 Konfigurasi <i>database server</i>	54
Gambar 4.7 <i>Background service script crawler</i>	54
Gambar 4.8 <i>User baru database</i>	56
Gambar 4.9 Struktur direktori	57
Gambar 4.10 Kode untuk membaca <i>config file crawler</i>	58
Gambar 4.11 Struktur <i>file TF IDF</i>	59
Gambar 4.12 Desain <i>database TF IDF</i> dan <i>PageRank</i>	60
Gambar 4.13 Potongan kode <i>TF IDF</i>	60
Gambar 4.14 Struktur <i>file PageRank</i>	61
Gambar 4.15 Potongan kode <i>PageRank</i>	61
Gambar 4.16 <i>Background service script PageRank</i>	62
Gambar 4.17 <i>Background service script TF IDF</i>	63
Gambar 4.18 Konfigurasi <i>background service</i> di <i>server</i>	63
Gambar 4.19 <i>REST API crawling extraction</i>	64
Gambar 4.20 <i>REST API crawling insertion</i>	65
Gambar 4.21 <i>REST API ranking TF IDF</i>	65
Gambar 4.22 <i>REST API ranking PageRank</i>	66
Gambar 4.23 <i>REST API</i> untuk menjalankan <i>crawler</i>	67
Gambar 4.24 Percobaan <i>keyword</i> memakai spasi pada <i>URL</i>	67
Gambar 4.25 Ukuran halaman di <i>database</i>	68
Gambar 4.26 Potongan kode <i>similarity score</i>	69
Gambar 4.27 <i>REST API similarity score</i>	69
Gambar 4.28 <i>REST API similarity score</i> dengan urutan berdasarkan <i>PageRank</i>	71
Gambar 4.29 Program <i>search engine</i> berbasis <i>console</i>	71
Gambar 4.30 Tampilan sederhana web <i>search engine</i>	72

Gambar 4.31 Struktur direktori dan <i>file</i>	73
Gambar 4.32 Contoh halaman dengan arah <i>link</i>	75
Gambar 4.33 <i>Library Python</i>	78
Gambar 4.34 <i>Grafik penggunaan RAM pada Crawler</i>	81
Gambar 4.35 Grafik penggunaan <i>RAM</i> pada <i>TF IDF</i>	82
Gambar 4.36 Grafik penggunaan <i>RAM</i> pada <i>PageRank</i>	83
Gambar 4.37 Desain <i>database</i> sistem	85
Gambar 4.38 <i>Class diagram</i> sistem	85
Gambar 4.39 <i>API</i> untuk menjalankan <i>crawling</i>	87
Gambar 4.40 <i>API</i> untuk berhenti <i>crawling</i>	87
Gambar 4.41 <i>API</i> untuk memulai penambahan halaman	88
Gambar 4.42 <i>API</i> untuk penambahan halaman	89
Gambar 4.43 <i>API</i> untuk mendapatkan informasi halaman berdasarkan jumlah	90
Gambar 4.44 <i>API</i> untuk mendapatkan informasi halaman berdasarkan <i>ID</i> .	91
Gambar 4.45 <i>API</i> untuk mendapatkan <i>ranking TF IDF</i>	92
Gambar 4.46 <i>API</i> untuk mendapatkan <i>ranking PageRank</i>	93
Gambar 4.47 <i>API</i> untuk mendapatkan <i>ranking</i> secara keseluruhan	94
Gambar 4.48 Program <i>search engine</i> berbasis <i>console</i>	95
Gambar 4.49 Tampilan web sederhana <i>search engine</i>	95
Gambar 4.50 <i>Source code</i> dan dokumentasi di Github	96

DAFTAR TABEL

Tabel 3.1	<i>Product Backlog</i>	39
Tabel 3.2	<i>Sprint 1 Backlog</i>	41
Tabel 3.3	<i>Functional test case</i>	48
Tabel 3.4	<i>Non-functional test case</i>	49
Tabel 4.1	<i>Sprint 2 Backlog</i>	51
Tabel 4.2	<i>Sprint 3 Backlog</i>	55
Tabel 4.3	Referensi kode dan <i>library</i>	58
Tabel 4.4	<i>Sprint 4 Backlog</i>	59
Tabel 4.5	<i>Sprint 5 Backlog</i>	62
Tabel 4.6	<i>Sprint 6 Backlog</i>	64
Tabel 4.7	<i>Sprint 7 Backlog</i>	66
Tabel 4.8	<i>Sprint 8 Backlog</i>	68
Tabel 4.9	<i>Sprint 9 Backlog</i>	70
Tabel 4.10	Contoh halaman dengan isi konten	75
Tabel 4.11	Hasil relevansi dari <i>query</i> "mamalia adalah"	77
Tabel 4.12	<i>Functional Testing</i>	80
Tabel 4.13	Perbandingan <i>response time API</i>	84
Tabel 4.14	Uji keandalan <i>server</i>	84
Tabel 4.15	<i>Routing table REST API</i>	86

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 antrean 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.

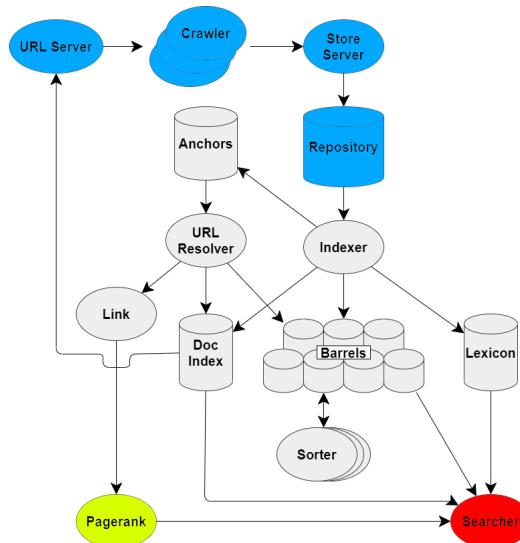
Dalam pengembangan *search engine*, selain perangkingan situs web diperlukan sebuah metode *document ranking*, yaitu untuk memverifikasi apakah konten dalam situs web tersebut relevan atau tidak. Salah satu metode yang paling umum digunakan adalah metode *TF-IDF*. *TF-IDF* atau *Term Frequency-Inversed 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 yang menggunakan *web crawler*, *page ranking*, dan *document ranking* berbasis *TF IDF*?”

C. Pembatasan Masalah

Pembatasan masalah pada penelitian ini antara lain:

1. Arsitektur *search engine* yang dibuat mencakup struktur *project*, rancangan diagram, konfigurasi *server*, *web service*, program *console*, serta dokumentasi lengkap *project*.
2. Mengombinasikan metode relevansi pencarian berbasis *TF IDF* dan *Google PageRank*.
3. Pengujian arsitektur menggunakan teknik *functional* dan *non-functional testing*.

D. Tujuan Penelitian

1. Membuat arsitektur 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 *Veronica*. 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 apa pun 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 *Nochtern Light*. *Yahoo!* dianggap salah satu yang terfavorit tetapi karena fungsi pencarinya 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.

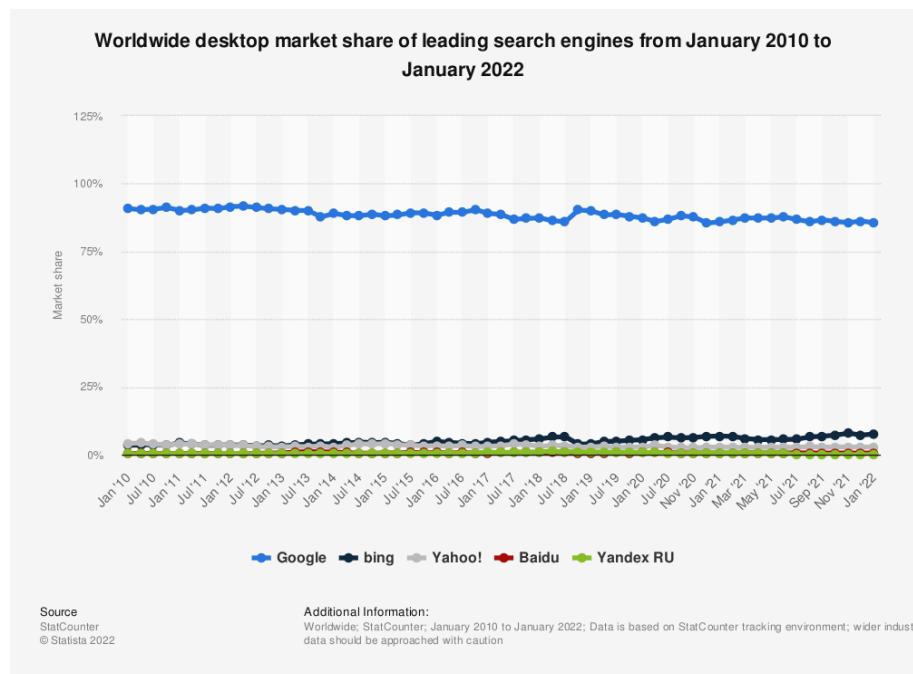
Dan pada tahun 1998, lahir satu *search engine* yang di kemudian hari menjadi *search engine* yang paling terkenal di seluruh negara. *Google* diciptakan oleh Sergey Brin dan Larry Page yang merupakan mahasiswa Stanford California. Cara mereka untuk meningkatkan kualitas pencarian agar lebih akurat adalah 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*.

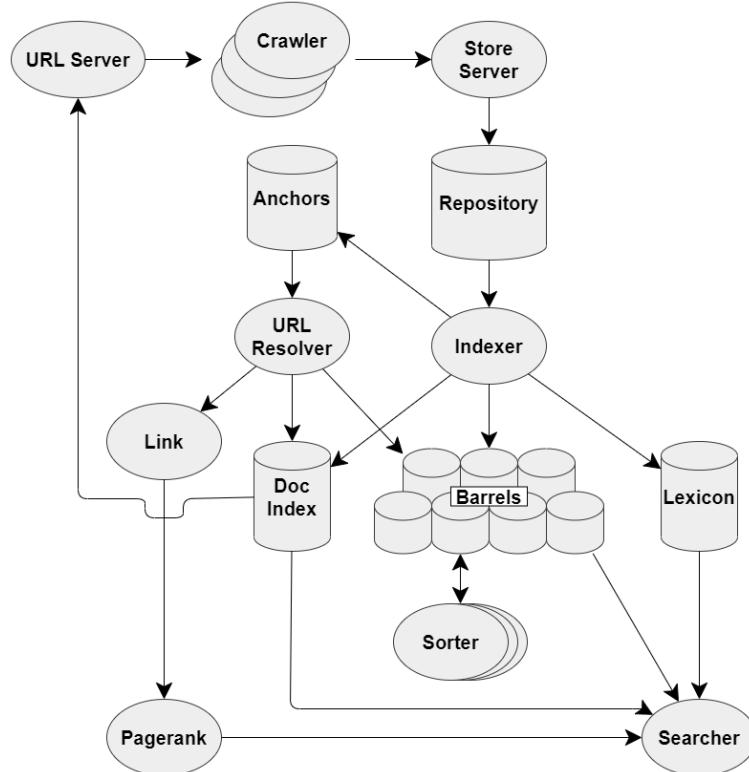


Gambar 2.1: *Market share search engine* tahun 2010 hingga 2022 (Statista, 2022)

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 *anchor file* dan mengubah *URL* yang sebelumnya relatif menjadi absolut, serta mengubahnya 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). *Web crawler* akan mengindeks dan mengunduh semua konten dari internet, lalu menyimpannya ke dalam *database*. *Web crawler* juga biasa dikenal sebagai *web spider*, *web bot*, dan *bot crawler*.

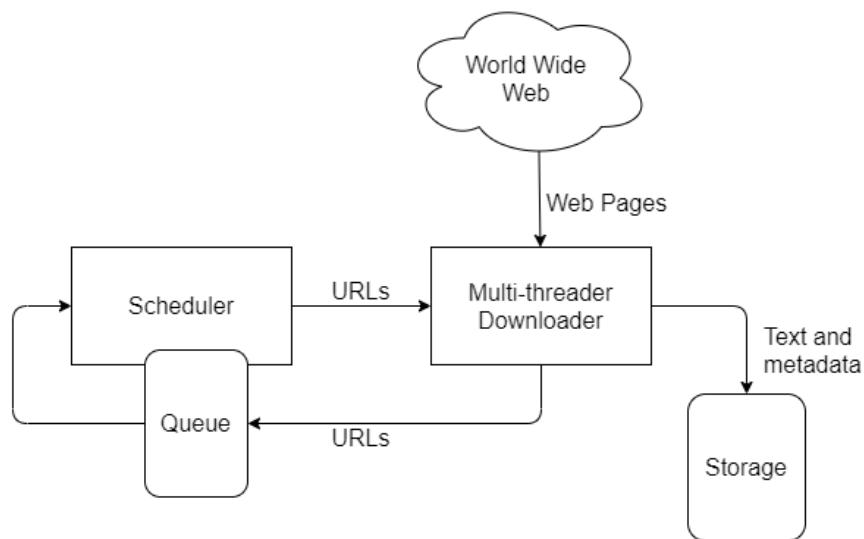
Secara kasar, *crawler* dimulai dari *URL* halaman awal P_0 . Kemudian *crawler* merayap ke P_0 , mengekstrak semua *URL* yang terdapat di dalam P_0 , dan menambahkannya ke antrean *URL* untuk dipindai. Setelah itu, *crawler* mengambil *URL* dari antrean (dengan urutan tertentu), dan mengulangi prosesnya (Cho dkk., 1998).

Setiap mesin pencari memiliki *web crawler*-nya masing-masing. Misalnya, *Google* memiliki *crawler* bernama *Googlebot*, *Bing* memiliki *Bingbot*, *DuckDuckGo* memiliki *DuckDuckBot*, *Yahoo* memiliki *Slurp Bot*, dll.

Selain mengindeks konten yang ada di internet, *web crawler* juga dapat dimanfaatkan untuk beberapa hal seperti:

1. Membandingkan harga produk di internet
2. Membantu *data mining website*
3. Memberikan data untuk performa suatu *website*

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 dalam beberapa minggu.

High level architecture of web crawler ditunjukkan pada Gambar 2.3, di mana arsitektur ini membutuhkan *scheduler* dan *multi-threader downloader* dengan dua struktur data utamanya yaitu *web page (text)* dan *URL queue*.

Gambar 2.3 juga menunjukkan bahwa proses *web crawler* dimulai dari beberapa web yang akan diunduh oleh *multi-threader downloader*, lalu *downloader* ini akan menyimpan konten web sekaligus mengambil semua *URL* yang terdapat pada web tersebut dan menyimpannya ke dalam antrean.

F. Algoritma *Crawling*

Secara umum, algoritma yang digunakan untuk *crawling* menggunakan algoritma *breadth first search*. Pada penerapannya dalam graf, algoritma *breadth first search* ini akan melakukan pencarian yang dimulai dari pemilihan *node* awal kemudian dilanjutkan dengan pencarian bertahap *level* demi *level*. Algoritma ini merupakan bentuk paling sederhana dari algoritma *crawling*.

Pada proses *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, \dots, p_n starting URLs

```

1:  $Q = p_1, p_2, \dots, p_n$ , 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, \dots, 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.

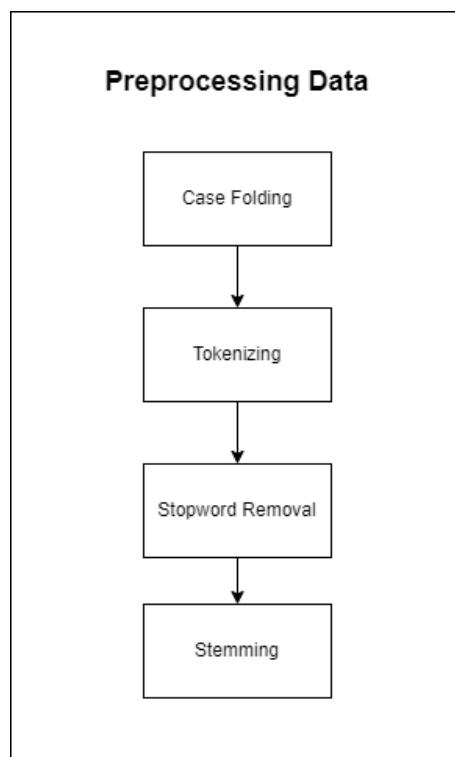
G. Preprocessing Data

Setiap dokumen atau data sering kali 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 ke dalam 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 *Term Frequency-Inverse Document Frequency*

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)*:

$$tf_{t,d} = \frac{\text{Jumlah kemunculan kata } t \text{ pada dokumen } d}{\text{Jumlah semua kata yang ada di dokumen } d} \quad (2.1)$$

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*, maka semakin tidak penting kata tersebut, dan sebaliknya. Berikut rumus dari *Inverse Document Frequency (IDF)*:

$$idf_t = \log\left(\frac{\text{Jumlah dokumen}}{\text{Jumlah dokumen yang mengandung kata } t}\right) \quad (2.2)$$

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).

$$tfidf_{t,d} = tf_{t,d} \times idf_t \quad (2.3)$$

Setelah mempunyai nilai *TF* dan *IDF*, nilai bobot pada hubungan antara kata *t* dengan dokumen *d* (nilai *TF IDF*) adalah hasil perkalian dari nilai *TF* dan *IDF*.

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 mengeklik *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.

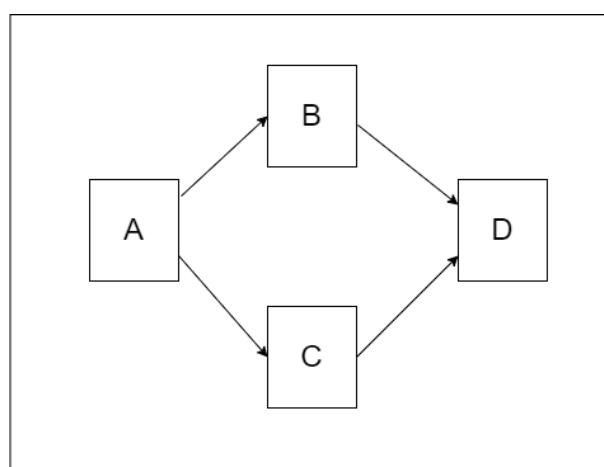
Meskipun *PageRank* secara populer dikaitkan dengan Larry Page dan Sergey Brin, pendiri *Google*, versi awal masalah *eigenvalue* dieksplorasi pada tahun 1976 oleh Gabriel Pinski dan Francis Narin dalam upaya mereka untuk menentukan peringkat jurnal ilmiah. Thomas Saaty mengeksplorasi masalah yang sama 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 mengonseptualisasikan 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):

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

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):

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

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 mengajinya 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)

N. *Cosine Similarity*

Metode *cosine similarity* merupakan metode yang digunakan untuk menghitung tingkat kesamaan (*similarity*) antar dua buah objek yang dinyatakan dalam dua buah vektor dengan menggunakan kata kunci (*keyword*) dari sebuah dokumen sebagai ukuran.

Dalam *search engine*, metode ini dapat digunakan untuk sebagai ukuran kemiripan antara kueri dari pengguna dengan masing-masing dokumen yang tersimpan. Setelah itu, skor kemiripan tersebut dapat ditambahkan dengan skor *PageRank* sehingga mendapatkan hasil yang lebih relevan (Roul dkk., 2019).

Pada pengukuran similaritas dengan *cosine similarity*, nilai *TF IDF* merupakan nilai setiap dimensi dokumen yang akan dibandingkan. Nilai *cosine similarity* antara 2 buah vektor dokumen A dan B dapat didefinisikan sebagai berikut:

$$Sim_{A,B} = \frac{AxB}{|A||B|} = \frac{\sum_{i=1}^n A_i x B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} x \sqrt{\sum_{i=1}^n (B_i)^2}} \quad (2.6)$$

Pada persamaan di atas, AxB merupakan *dot product* antara dokumen A dan B, sedangkan $|A||B|$ merupakan perkalian panjang vektor antara dokumen A dan B. *Dot product* dilakukan dengan melakukan perkalian biasa antara nilai *TF IDF* dokumen A dengan dokumen B untuk setiap dimensi. Sedangkan panjang vektor A dan B, dapat diketahui dengan cara menghitung akar kuadrat dari penjumlahan pangkat dua masing-masing nilai *TF IDF* pada setiap dokumen A dan B (Firdaus dkk., 2019).

BAB III

METODOLOGI PENELITIAN

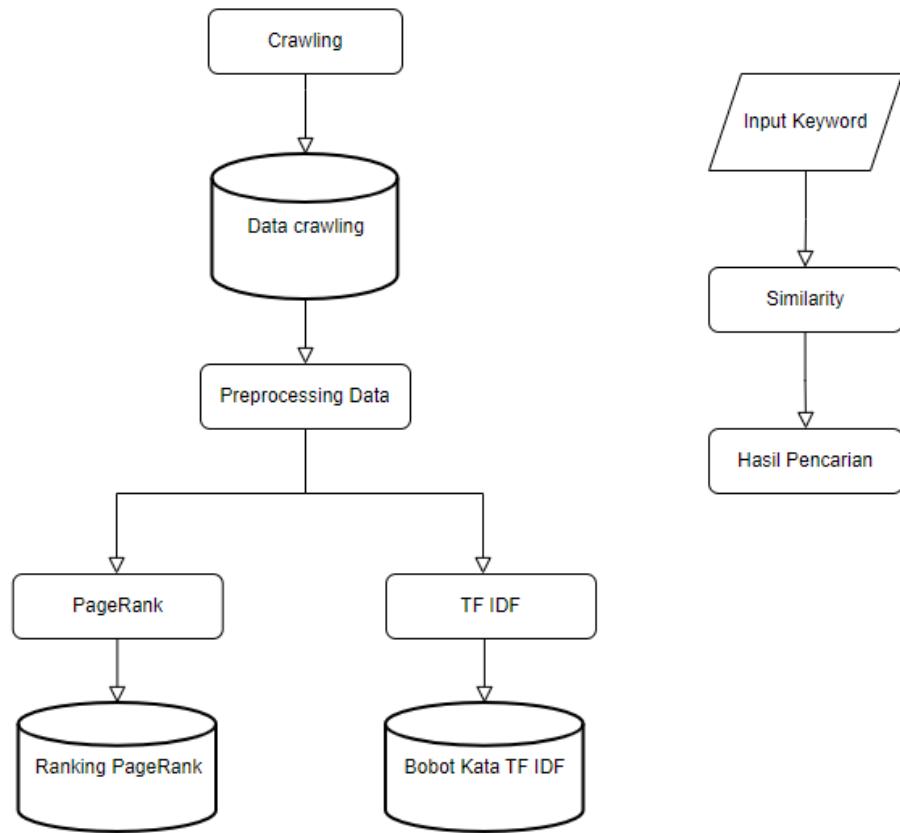
A. Deskripsi Sistem



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 peringkiran *website*. Selain itu, proses pengembangan sistem ini memakai metode pengembangan Scrum.

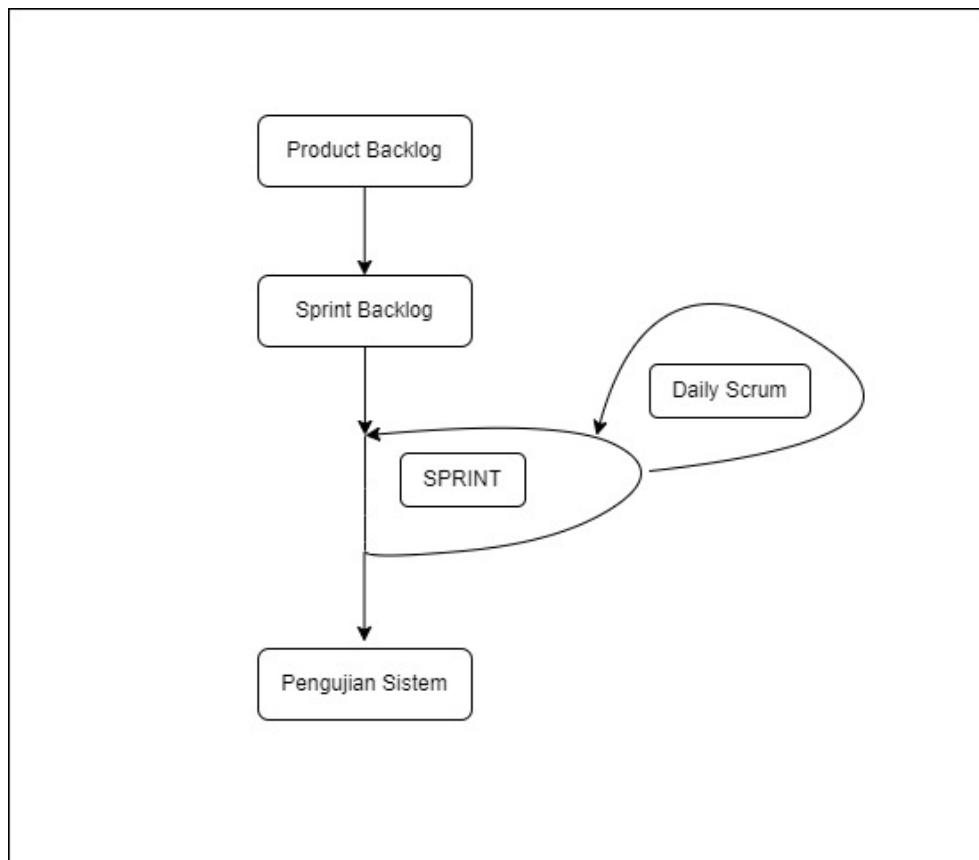


Gambar 3.2: Flowchart Sistem

Proses dari sistem *search engine* dimulai dengan mengumpulkan data ke dalam *database* sebanyak-banyaknya menggunakan *crawler* dan data tersebut akan diolah melalui proses *preprocessing data*, *PageRank*, dan *TF IDF*. Setelah data sudah siap, maka sistem dimulai dengan meminta *input* berupa *keyword* dari pengguna lalu *keyword* tersebut akan diproses melalui proses *similarity*, yaitu proses mengambil *website* yang relevan dengan cara menghitung *cosine similarity* berdasarkan bobot *TF IDF* yang ada lalu dijumlahkan dengan nilai *PageRank*. Setelah proses *similarity* selesai, 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**.

Tabel 3.1: Product Backlog

No	Story	Sprint	Status	Priority
1	Ubah struktur project program <i>crawler</i> menjadi modular	1	Completed	Penting
2	Konfigurasi program <i>crawler</i> sebagai <i>background service</i> di sistem operasi	2		Penting
3	Perancangan struktur <i>project</i> arsitektur <i>search engine</i>	3		Penting
4	Perancangan struktur <i>package</i> , <i>class</i> , dan fungsi program <i>TF IDF</i>	4		Penting
5	Perancangan struktur <i>package</i> , <i>class</i> , dan fungsi program <i>Page Rank</i>	5		Penting
6	Konfigurasi program <i>page rank</i> dan <i>TF IDF</i> sebagai <i>background service</i> di sistem operasi	6		Penting
7	Perancangan dan implementasi <i>REST API</i>	7		Penting
8	Integrasi <i>PageRank</i> dan <i>TF IDF</i> untuk menghasilkan <i>similarity score</i>	8		Penting
9	Buat program berbasis <i>console</i> dan tampilan web sederhana untuk melakukan pencarian	9		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	Ubah struktur kode program <i>crawler</i> menjadi modular	Membuat rancangan <i>class diagram</i>	Completed
2		Membuat rancangan <i>entity relationship diagram</i> untuk <i>database</i>	Completed
3		Mengubah struktur kode <i>crawler</i> sesuai dengan rancangan <i>class diagram</i>	Completed
4		Implementasi <i>threading</i> untuk menjalankan lebih dari satu proses dalam satu waktu	Completed
5		<i>Testing</i> program versi terbaru	Completed

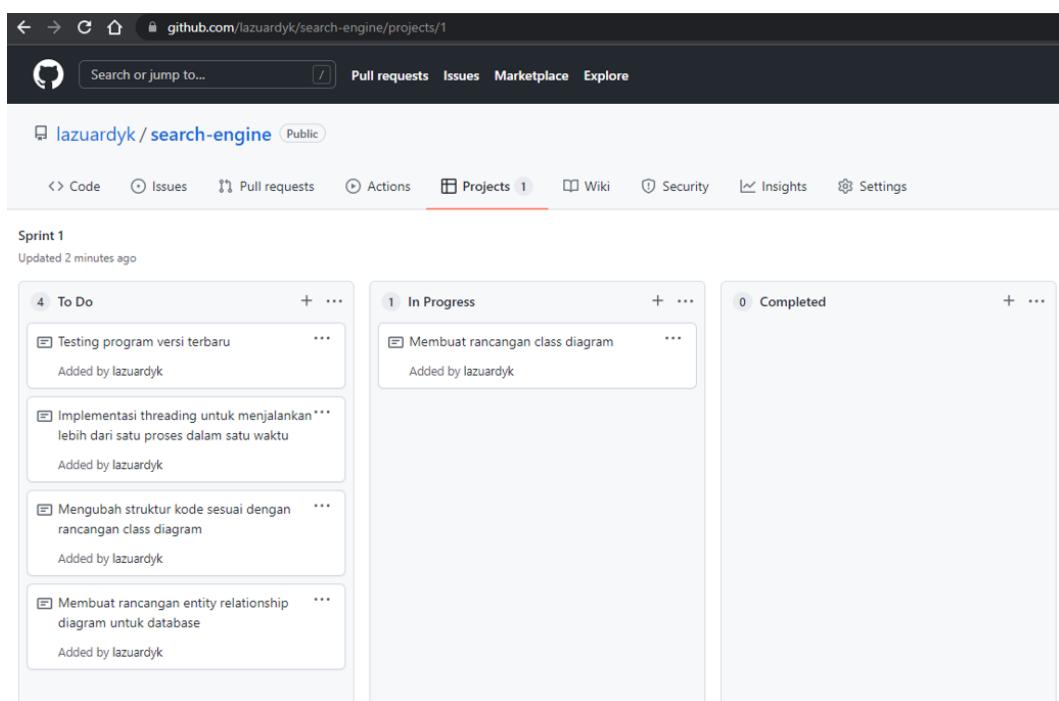
3. Daily Scrum, Sprint Review, dan Sprint Retrospective

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 penggerjaan 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.

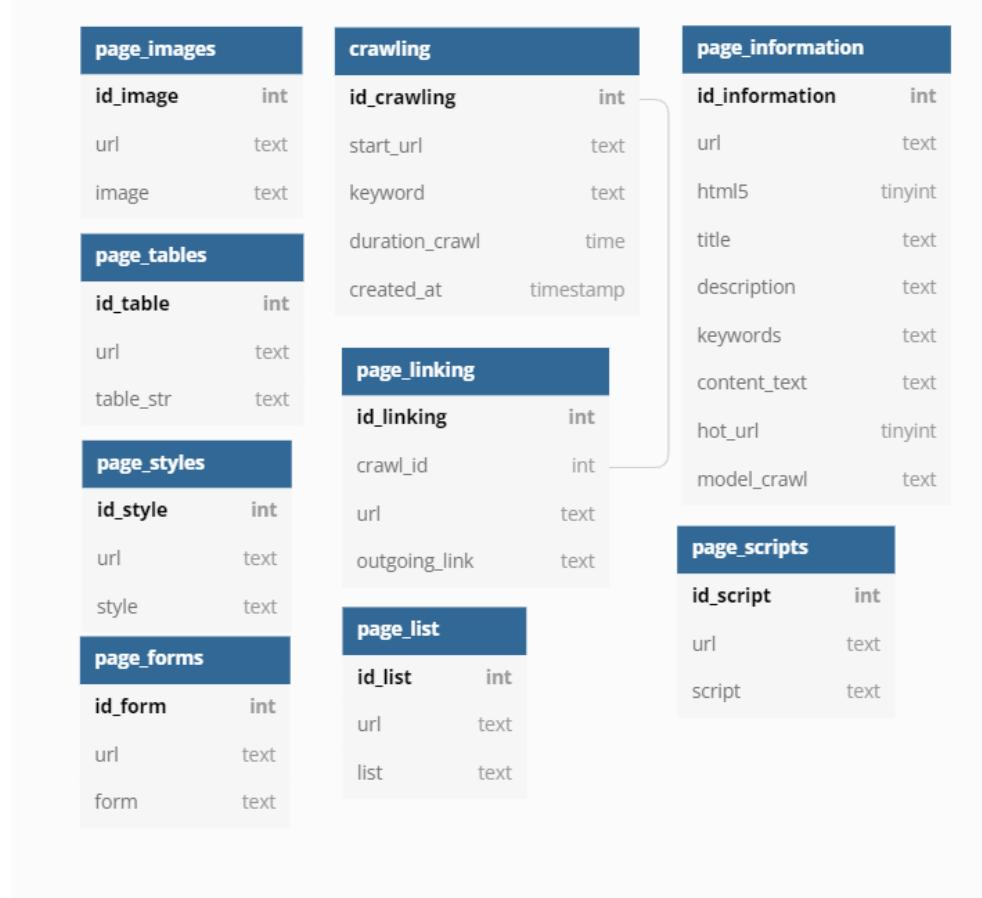


Gambar 3.4: *GitHub Projects Sprint-1*

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 penggerjaan 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*.

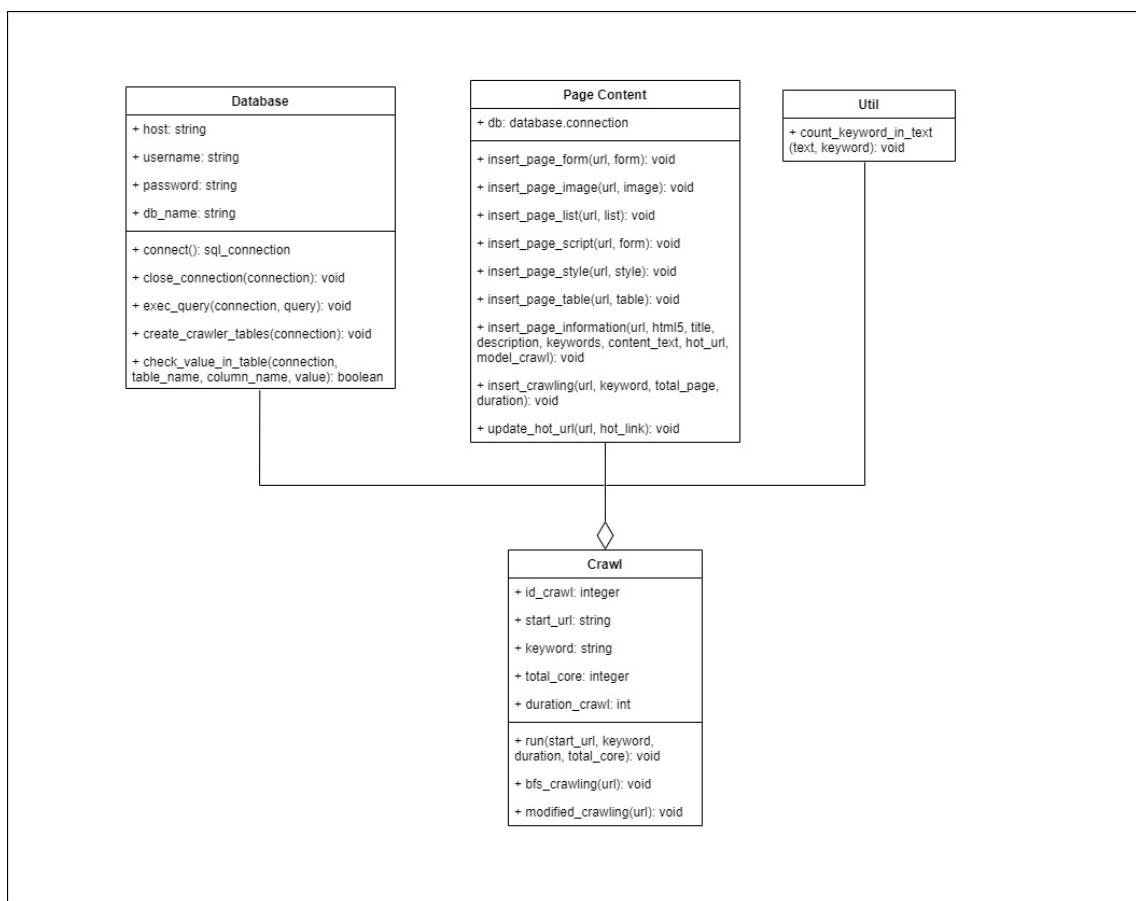


Gambar 3.5: Entity Relationship Diagram Sprint-1

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

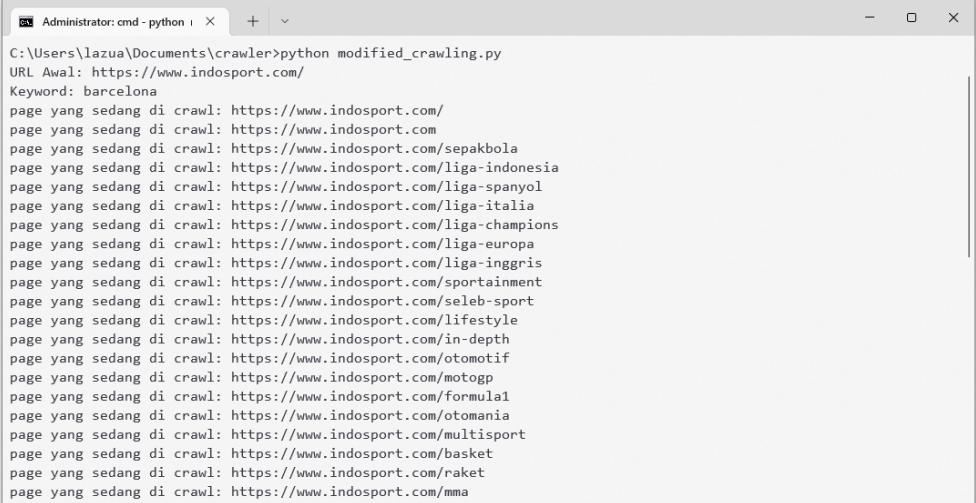
page content yang berisi fungsi-fungsi untuk menyimpan konten dari halaman *website*, kelas *util* yang terdapat fungsi-fungsi pendukung, dan kelas *crawl* sebagai kelas utama untuk menjalankan *crawler*.

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*.

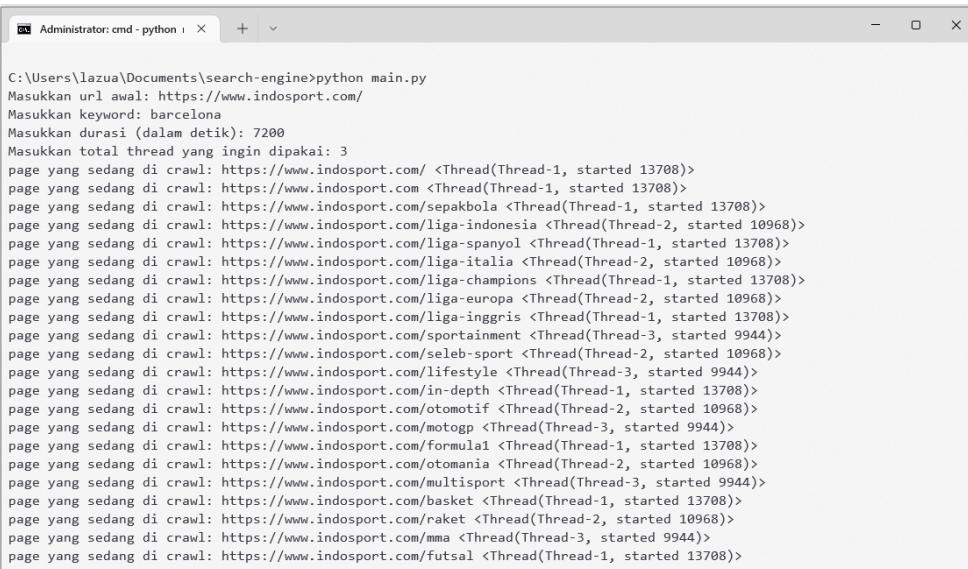


```

Administrator:cmd - python i  x  +  v
C:\Users\lazua\Documents\crawler>python modified_crawling.py
URL Awal: https://www.indosport.com/
Keyword: barcelona
page yang sedang di crawl: https://www.indosport.com/
page yang sedang di crawl: https://www.indosport.com/
page yang sedang di crawl: https://www.indosport.com/sepakbola
page yang sedang di crawl: https://www.indosport.com/liga-indonesia
page yang sedang di crawl: https://www.indosport.com/liga-spanyol
page yang sedang di crawl: https://www.indosport.com/liga-italia
page yang sedang di crawl: https://www.indosport.com/liga-champions
page yang sedang di crawl: https://www.indosport.com/liga-europa
page yang sedang di crawl: https://www.indosport.com/liga-inggris
page yang sedang di crawl: https://www.indosport.com/sporentainment
page yang sedang di crawl: https://www.indosport.com/seleb-sport
page yang sedang di crawl: https://www.indosport.com/lifestyle
page yang sedang di crawl: https://www.indosport.com/in-depth
page yang sedang di crawl: https://www.indosport.com/otomotif
page yang sedang di crawl: https://www.indosport.com/motogp
page yang sedang di crawl: https://www.indosport.com/formula1
page yang sedang di crawl: https://www.indosport.com/otomania
page yang sedang di crawl: https://www.indosport.com/multisport
page yang sedang di crawl: https://www.indosport.com/basket
page yang sedang di crawl: https://www.indosport.com/raket
page yang sedang di crawl: https://www.indosport.com/mma

```

Gambar 3.7: Pengujian *crawler* penelitian sebelumnya



```

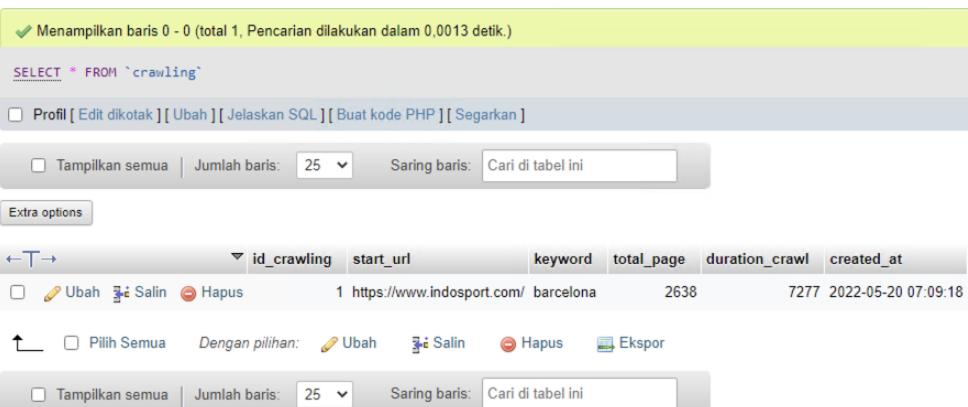
Administrator: cmd - python i + v

C:\Users\lazua\Documents\search-engine>python main.py
Masukkan url awal: https://www.indosport.com/
Masukkan keyword: barcelona
Masukkan durasi (dalam detik): 7200
Masukkan total thread yang ingin dipakai: 3
page yang sedang di crawl: https://www.indosport.com/ <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/ <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/sepakbola <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/liga-indonesia <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/liga-spanyol <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/liga-italia <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/liga-champions <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/liga-europa <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/liga-inggris <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/sportainment <Thread(Thread-3, started 9944)>
page yang sedang di crawl: https://www.indosport.com/seleb-sport <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/lifestyle <Thread(Thread-3, started 9944)>
page yang sedang di crawl: https://www.indosport.com/in-depth <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/otomotif <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/motogp <Thread(Thread-3, started 9944)>
page yang sedang di crawl: https://www.indosport.com/formula1 <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/otomotif <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/multisport <Thread(Thread-3, started 9944)>
page yang sedang di crawl: https://www.indosport.com/basket <Thread(Thread-1, started 13708)>
page yang sedang di crawl: https://www.indosport.com/raket <Thread(Thread-2, started 10968)>
page yang sedang di crawl: https://www.indosport.com/mma <Thread(Thread-3, started 9944)>
page yang sedang di crawl: https://www.indosport.com/futsal <Thread(Thread-1, started 13708)>

```

Gambar 3.8: Pengujian *crawler* versi terbaru

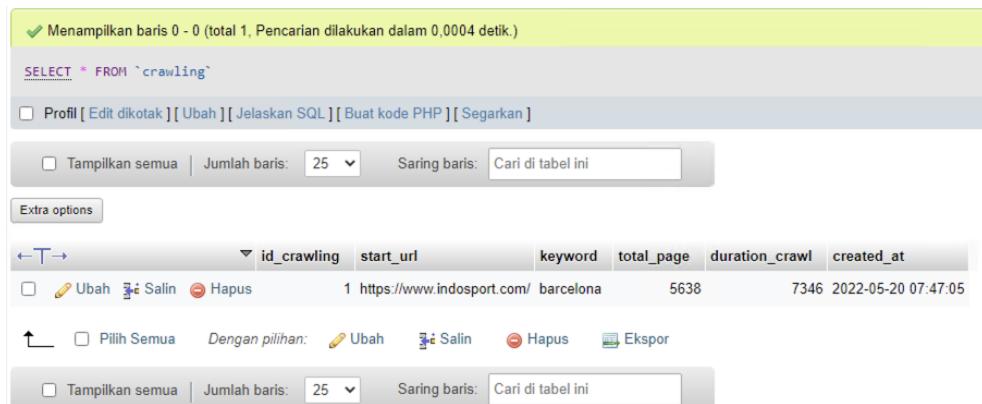
Gambar 3.7 dan gambar 3.8 merupakan tampilan *crawler* versi lama dan terbaru yang berjalan setelah dimasukkan *input* dari *user*. Hasil dari kedua *crawler* yang telah selesai dijalankan ini terdapat pada gambar 3.9 dan gambar 3.10.



Menampilkan baris 0 - 0 (total 1, Pencarian dilakukan dalam 0,0013 detik.)

SELECT * FROM `crawling`						
<input type="checkbox"/> Profil [Edit dikotak] [Ubah] [Jelaskan SQL] [Buat kode PHP] [Segarkan]						
<input type="checkbox"/> Tampilkan semua		Jumlah baris:	25	Saring baris:	Cari di tabel ini	
<input type="checkbox"/> Extra options						
← → ↓ id_crawling start_url keyword total_page duration_crawl created_at						
<input type="checkbox"/> Ubah		<input type="checkbox"/> Salin	<input type="checkbox"/> Hapus	1 https://www.indosport.com/ barcelona 2638 7277 2022-05-20 07:09:18		
↑ <input type="checkbox"/> Pilih Semua Dengan pilihan: Ubah <input type="checkbox"/> Salin <input type="checkbox"/> Hapus Eksport						
<input type="checkbox"/> Tampilkan semua						

Gambar 3.9: Hasil pengujian *crawler* penelitian sebelumnya



Menampilkan baris 0 - 0 (total 1, Pencarian dilakukan dalam 0.0004 detik.)

```
SELECT * FROM `crawling`
```

Profil [Edit dikotak] [Ubah] [Jelaskan SQL] [Buat kode PHP] [Segarkan]

Tampilkan semua | Jumlah baris: 25 | Saring baris: Cari di tabel ini

Extra options

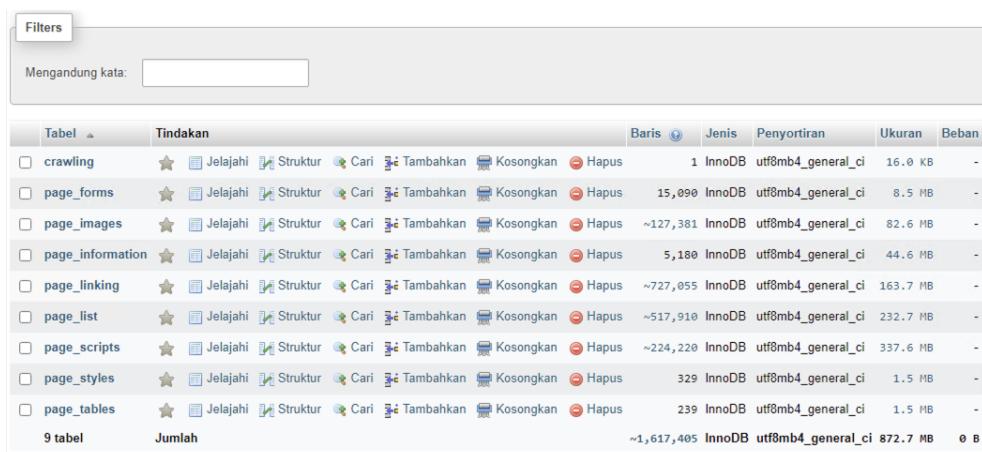
	id_crawling	start_url	keyword	total_page	duration_crawl	created_at
<input type="checkbox"/>	Ubah Salin Hapus	1 https://www.indosport.com/ barcelona		5638	7346	2022-05-20 07:47:05

Pilih Semua | Dengan pilihan: [Ubah](#) [Salin](#) [Hapus](#) [Eksport](#)

Tampilkan semua | Jumlah baris: 25 | Saring baris: Cari di tabel ini

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*.

E. Pengujian Sistem

Pengujian sistem dilakukan menggunakan teknik *functional* dan *non-functional testing*. *Functional testing* adalah pengujian yang memfokuskan kepada fungsi sistem dan untuk memverifikasi setiap fitur dari sistem bekerja dengan baik dan sudah sesuai dengan *requirements*. Serangkaian kasus uji atau *test case* yang akan dilakukan pada *functional testing* adalah sebagai berikut.

Tabel 3.3: Functional test case

No	Skenario Pengujian	Yang Diharapkan
1	Menjalankan program <i>crawler</i> , <i>page rank</i> dan <i>TF IDF</i> di lokal komputer dan melihat hasilnya di <i>database</i>	Data tersimpan dengan baik di <i>database</i>
2	Melakukan <i>start</i> dan <i>stop</i> pada setiap <i>background service</i> di <i>server</i>	<i>Start</i> dan <i>stop</i> pada <i>background service</i> di <i>server</i> dapat bekerja dengan baik
3	Menjalankan semua <i>routes</i> yang ada pada <i>REST API</i> dengan <i>Postman</i>	Kode dan isi <i>response</i> tiap <i>endpoint</i> benar dan sesuai fungsinya
4	Melakukan pencarian pada tampilan web dan program <i>console search engine</i>	Mempunyai hasil pencarian yang relevan

Sedangkan *non-functional testing* adalah pengujian yang dilakukan kepada hal-hal yang tidak berhubungan dengan fungsi utama sistem seperti performa, keamanan, dan efisiensi. Jenis *non-functional testing* yang akan dipakai pada arsitektur ini adalah *performance testing*, yaitu pengujian untuk menguji ketahanan dan kestabilan sistem. Serangkaian kasus uji atau *test case* yang akan dilakukan pada *non-functional testing* adalah sebagai berikut.

Tabel 3.4: *Non-functional test case*

No	Skenario Pengujian	Yang Diharapkan
1	Menjalankan tiap komponen pada <i>server</i> untuk melihat lama waktu dan besar penggunaan <i>RAM</i>	Lama waktu komponen <i>crawler</i> sesuai dengan durasi yang ditentukan dan penggunaan RAM tiap komponennya stabil dan tidak <i>overload</i>
2	Membandingkan <i>response time</i> tiap <i>endpoint API</i> pada saat <i>background service</i> berjalan dan pada saat <i>background service</i> tidak berjalan	Selisih <i>response time</i> -nya tidak terlalu jauh
3	Uji coba server dengan melakukan kunjungan ke salah satu API dengan jumlah permintaan 10, 1000, dan 10000	<i>Response</i> pada <i>API</i> tetap keluar dan tidak terjadi <i>error</i> atau <i>timeout</i>

BAB IV

Hasil dan Pembahasan

A. Perancangan Sistem

Arsitektur *search engine* yang dibuat terdiri dari struktur *project*, rancangan diagram, konfigurasi yang diperlukan untuk *server*, program *console* dan tampilan web sederhana, *web service* berupa *REST API* yang nantinya dapat digunakan untuk *frontend*, serta dokumentasi lengkap *project*.

Perancangan arsitektur *search engine* yang mengintegrasikan *web crawler*, algoritma *page ranking*, dan *document ranking* ini menggunakan metode Scrum. Dalam metodologi Scrum, pengembangan sistem dilakukan melalui iterasi pada setiap *sprint*, dan urutan pelaksanaan *sprint* terdapat pada *product backlog*.

Bahasa pemrograman yang digunakan dalam sistem ini adalah *Python* dan *database* yang digunakan adalah *MySQL*. Proses pengembangan sistem melibatkan langkah-langkah berikut:

1. *Sprint reports*

Berikut merupakan *progress* iterasi dalam pelaksanaan *sprint* dari *sprint* kedua hingga *sprint* terakhir.

1. *Sprint-2* (16 Juli 2022 - 22 Juli 2022)

Tabel 4.1: *Sprint 2 Backlog*

No	Story	Task	Status
1	Konfigurasi program <i>crawler</i> sebagai <i>background service</i> di sistem operasi	Mengubah kode <i>crawler</i> agar memiliki fitur <i>start</i> , <i>stop</i> , dan <i>resume</i>	<i>Completed</i>
2		Restrukturisasi kode <i>crawler</i> agar metode <i>BFS</i> dan <i>MSB</i> terdapat di <i>package</i> yang terpisah	<i>Completed</i>
3		<i>Upload crawler</i> ke <i>server</i>	<i>Completed</i>
4		<i>Install MySQL</i> dan <i>set up database</i> di <i>server</i>	<i>Completed</i>
5		Buat <i>background service</i> <i>script</i> dan jalankan <i>crawler</i> di <i>server</i> dengan metode <i>BFS</i>	<i>Completed</i>

- Potongan kode untuk fitur *start* dan *resume* pada *crawler*

```

57  def run(self) -> int:
58      """
59      Fungsi utama yang berfungsi untuk menjalankan proses crawling.
60
61      Returns:
62      |   page_count (int): Jumlah halaman yang berhasil dicrawl.
63      """
64      self.url_queue = queue.Queue()
65      self.start_time: float = time.time()
66
67      db_connection = self.db.connect()
68      if self.status.lower() == "start":
69      |   self.db.truncate_tables()
70      self.visited_urls = self.crawl_utils.get_visited_urls(db_connection)
71      self.page_count_start = self.db.count_rows(db_connection, "page_information")
72

```

Gambar 4.1: Kode fitur *start* dan *resume crawler* 1

```

73     urls_string = ""
74     if len(self.visited_urls) < 1:
75         print("Starting the crawler from the start urls...")
76         for url in self.start_urls:
77             if self.crawl_utils.is_valid_url(url):
78                 self.url_queue.put(url)
79                 urls_string += url + " "
80     else:
81         print("Resuming the crawler from the last urls...")
82         last_urls = self.visited_urls[-3:]
83         for url in last_urls:
84             urls_string += url + " "
85             self.scrape_links_for_resume(last_urls)
86     urls_string = urls_string[0 : len(urls_string) - 1]

```

Gambar 4.2: Kode fitur *start* dan *resume crawler* 2

- Potongan kode untuk fitur *stop* pada *crawler*

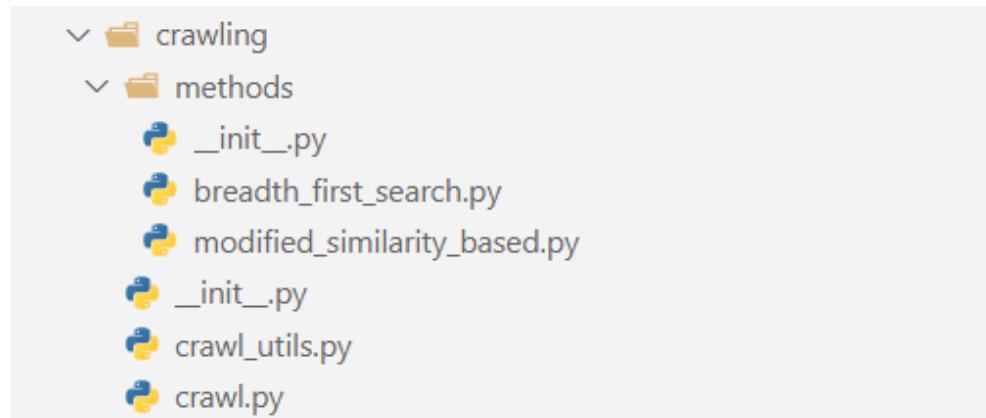
```

47     while True:
48         try:
49             time_now = time.time() - self.start_time
50             time_now_int = int(time_now)
51             if time_now_int >= self.duration_sec:
52                 print("Stopped because exceeded time limit...")
53                 break
54             target_url = self.url_queue.get(timeout=60)
55             if target_url not in self.visited_urls:
56                 self.visited_urls.append(target_url)
57                 futures.append(executor.submit(self.scrape_page, target_url))
58         except queue.Empty:
59             if self.crawl_utils.running_thread_count(futures) > 0:
60                 continue
61             else:
62                 print("Stopped because empty queue...")
63                 break
64         except KeyboardInterrupt:
65             print("Stopped because keyboard interrupt...")
66             break
67         except Exception as e:
68             print(e)
69             continue

```

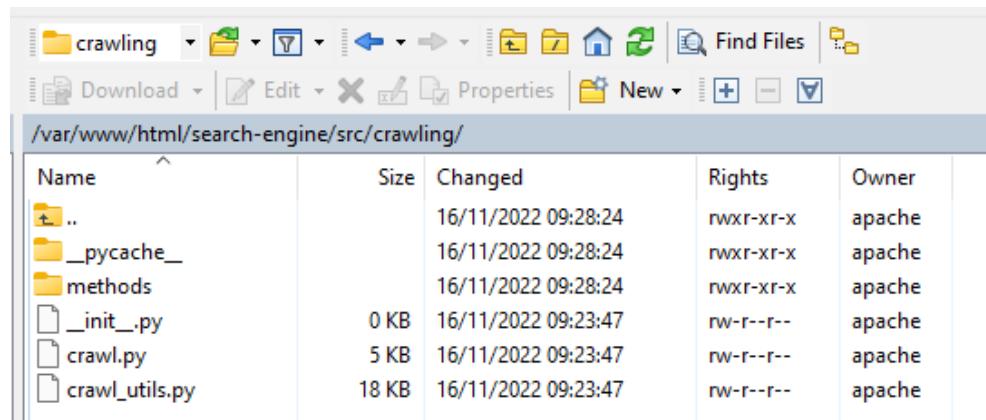
Gambar 4.3: Kode fitur *stop crawler*

- Struktur *file crawler* setelah metodenya dipisah dengan *package* yang berbeda



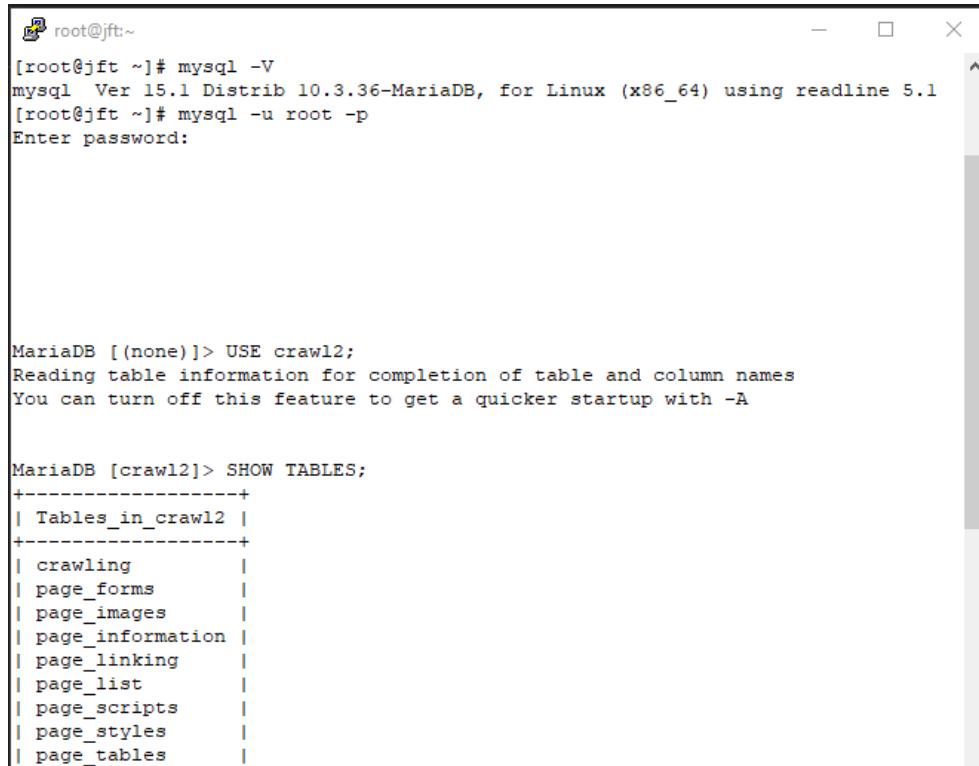
Gambar 4.4: Struktur *file crawler*

- *File crawler* yang sudah di-*upload* di *server*



Gambar 4.5: *File crawler* di *server*

- Konfigurasi *mysql* dan *set up database* pada *server*



```

root@jft:~#
[root@jft ~]# mysql -V
mysql  Ver 15.1 Distrib 10.3.36-MariaDB, for Linux (x86_64) using readline 5.1
[root@jft ~]# mysql -u root -p
Enter password:

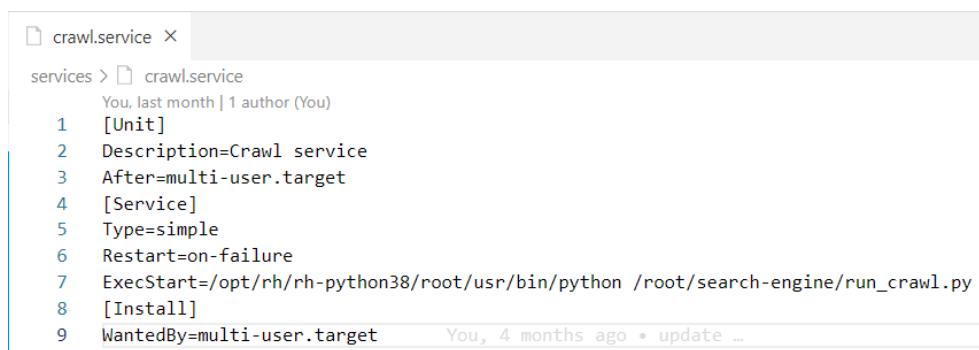
MariaDB [(none)]> USE crawl2;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

MariaDB [crawl2]> SHOW TABLES;
+-----+
| Tables_in_crawl2 |
+-----+
| crawling          |
| page_forms        |
| page_images       |
| page_information |
| page_linking     |
| page_list         |
| page_scripts      |
| page_styles       |
| page_tables       |
+-----+

```

Gambar 4.6: Konfigurasi *database server*

- *Background service script* untuk *crawler*



```

crawl.service x
services > crawl.service
You, last month | 1 author (You)
1 [Unit]
2 Description=Crawl service
3 After=multi-user.target
4 [Service]
5 Type=simple
6 Restart=on-failure
7 ExecStart=/opt/rh/rh-python38/root/usr/bin/python /root/search-engine/run_crawl.py
8 [Install]
9 WantedBy=multi-user.target
You, 4 months ago • update ...

```

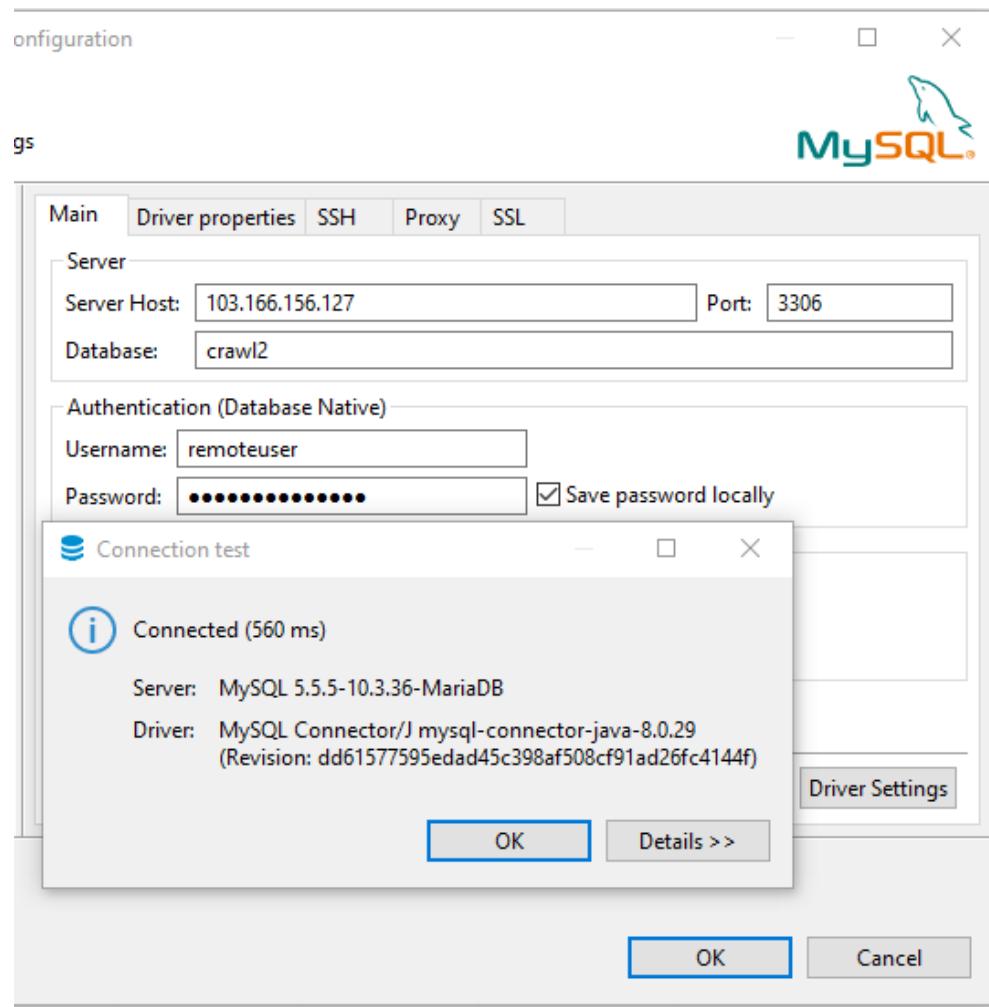
Gambar 4.7: *Background service script crawler*

2. *Sprint-3* (23 Juli 2022 - 29 Juli 2022)

Tabel 4.2: *Sprint 3 Backlog*

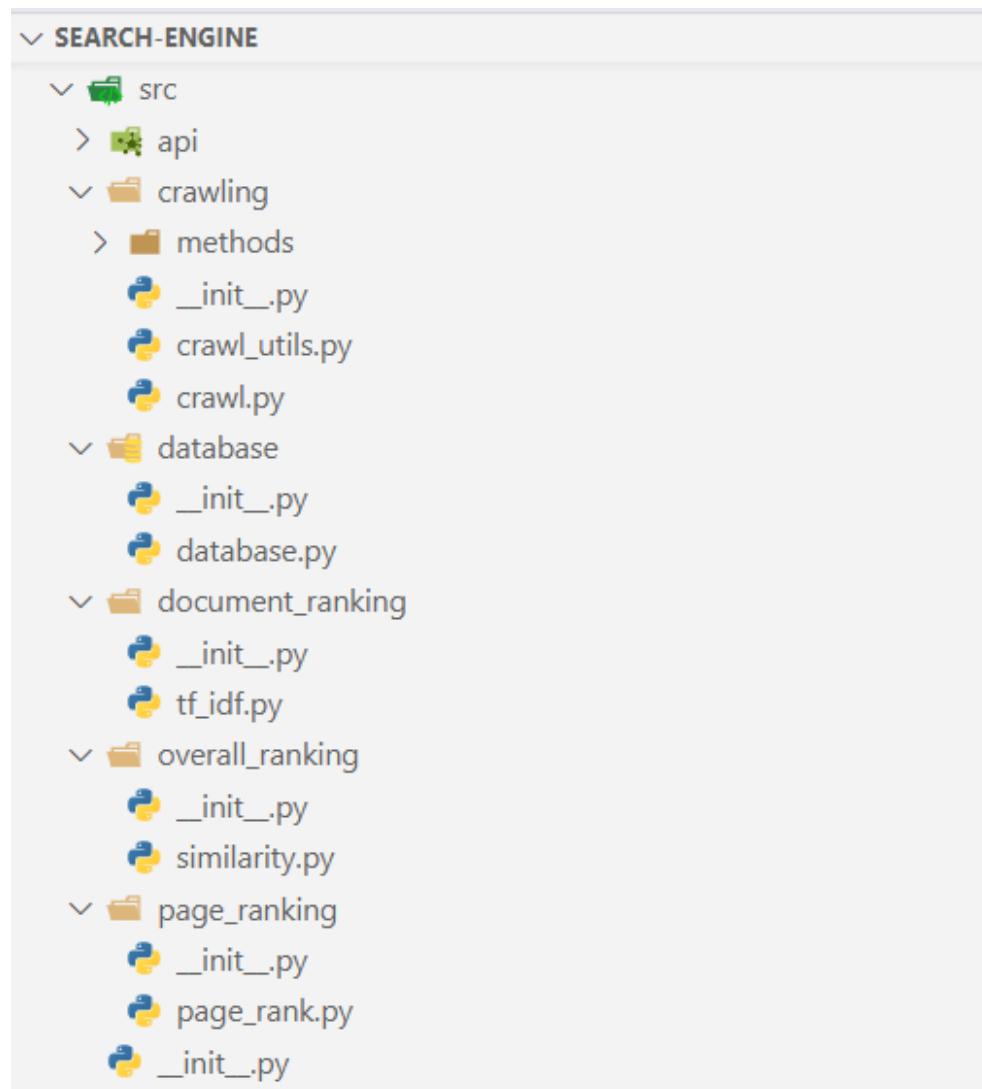
No	Story	Task	Status
1	Perancangan struktur <i>project</i> arsitektur <i>search engine</i>	Buat <i>user</i> baru di <i>database</i> pada <i>server</i> untuk akses dari luar <i>server</i>	Completed
2		Buat spesifikasi struktur direktori untuk <i>crawling</i> , <i>page ranking</i> , <i>document ranking</i> , dan <i>database</i>	Completed
3		Transformasi kode <i>crawling</i> sehingga dapat membaca parameter dari <i>config file</i> (<i>env</i>)	Completed
4		Cari dan pelajari <i>library</i> terkait dengan pengimplementasian <i>TF IDF</i> dan <i>PageRank</i>	Completed

- *User* baru pada *database server* dan *test* koneksi dari luar *server* dengan *software DBeaver*



Gambar 4.8: *User* baru *database*

- Struktur direktori untuk *crawling*, *page ranking*, *document ranking*, dan *database*



Gambar 4.9: Struktur direktori

- Potongan kode untuk membaca konfigurasi yang ada di *config file* pada *crawler*



```

run_crawl.py > ...
run_crawl.py > ...
6  if __name__ == "__main__":
7      load_dotenv()
8      db = Database()
9      db.create_tables()
10
11     start_urls = os.getenv("CRAWLER_START_URLS").split()
12     max_threads = os.getenv("CRAWLER_MAX_THREADS")
13     crawler_duration_sec = os.getenv("CRAWLER_DURATION_SECONDS")
14     try:
15         msb_keyword = os.getenv("CRAWLER_KEYWORD")
16     except:
17         msb_keyword = ""
18
19     if msb_keyword != "":
20         bfs_duration_sec = int(crawler_duration_sec) // 2
21         msb_duration_sec = int(crawler_duration_sec) // 2
22     else:
23         bfs_duration_sec = int(crawler_duration_sec)
24         msb_duration_sec = 0
25
26     c = Crawl(start_urls, max_threads, bfs_duration_sec, msb_duration_sec, msb_keyword)
27     c.run()
28

```

Gambar 4.10: Kode untuk membaca *config file crawler*

- Referensi kode dan *library* untuk pengimplementasian algoritma *TF IDF* dan *PageRank*

Tabel 4.3: Referensi kode dan *library*

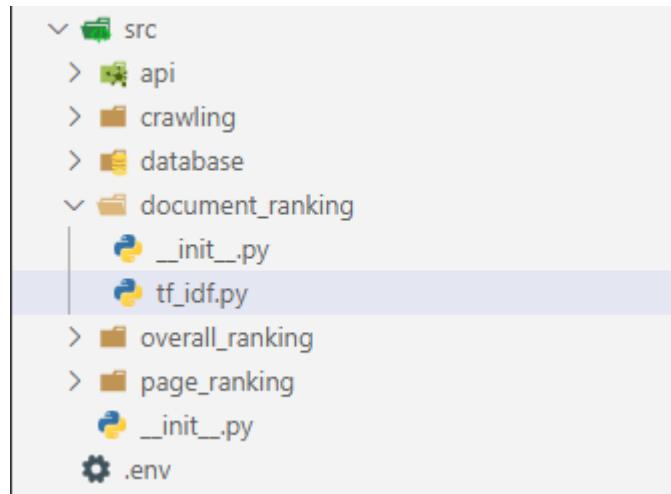
Kode	Referensi
<i>TF IDF</i>	https://www.kaggle.com/code/yclaudel/find-similar-articles-with-tf-idf
<i>PageRank</i>	https://github.com/nicholaskajoh/devsearch/blob/master/devsearch/pagerank.py

3. *Sprint-4* (30 Juli 2022 - 12 Agustus 2022)

Tabel 4.4: *Sprint 4 Backlog*

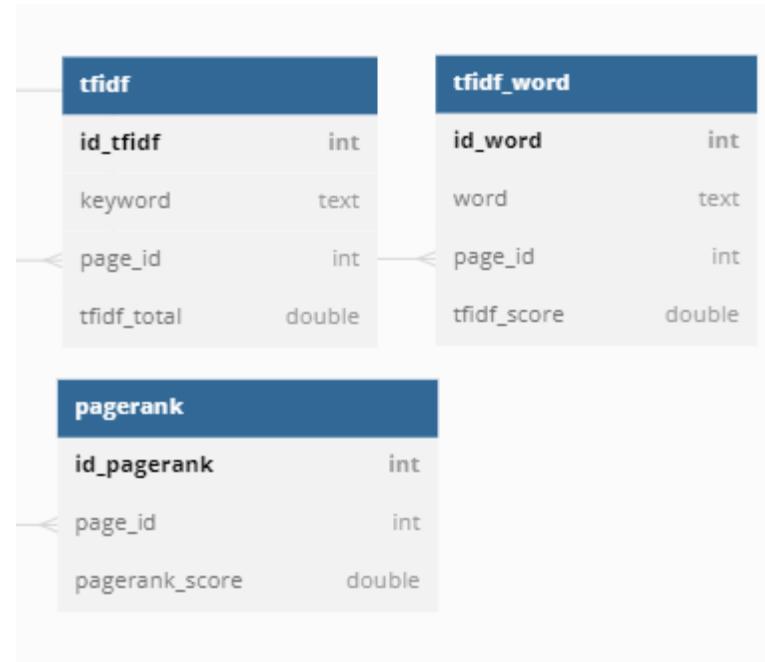
No	Story	Task	Status
1	Perancangan struktur <i>package, class, dan fungsi</i> program <i>TF IDF</i>	Buat spesifikasi struktur <i>class</i> dan fungsi untuk <i>TF IDF</i>	<i>Completed</i>
2		Desain <i>database</i> untuk <i>TF IDF</i>	<i>Completed</i>
3		Implementasi <i>TF IDF</i>	<i>Completed</i>
4	Perancangan struktur <i>package, class, dan fungsi</i> program <i>PageRank</i>	Buat spesifikasi struktur <i>class</i> dan fungsi untuk <i>PageRank</i>	<i>Completed</i>
5		Desain <i>database</i> untuk <i>PageRank</i>	<i>Completed</i>
6		Implementasi <i>PageRank</i>	<i>Completed</i>

- Struktur *file TF IDF*



Gambar 4.11: Struktur *file TF IDF*

- Desain *database TF IDF* dan *PageRank*



Gambar 4.12: Desain *database TF IDF* dan *PageRank*

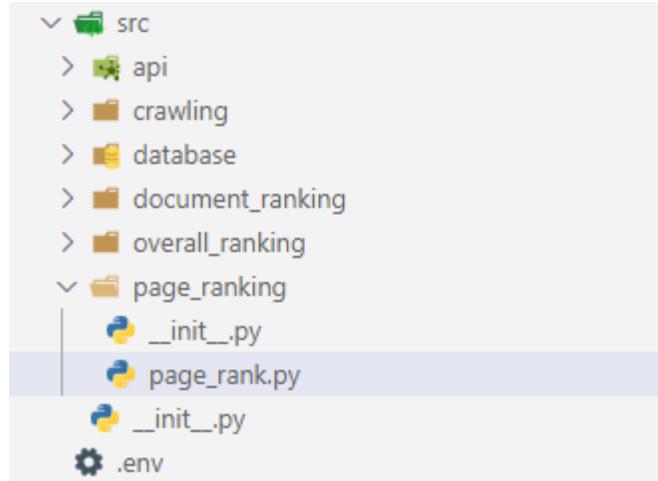
- Potongan kode pengimplementasian *TF IDF*

```

163     db_connection = self.db.connect()
164
165     # Ambil semua data halaman yang sudah di crawl ke dalam pandas dataframe
166     query = "SELECT * FROM `page_information`"
167     df = pd.read_sql(query, db_connection)
168     text_content = df['content_text'] # Konten teks dari halaman yang sudah dicrawl
169
170     # Buat model menggunakan TfidfVectorizer
171     vectorizer = TfidfVectorizer(
172         lowercase=True, # Untuk konversi ke lower case
173         use_idf=True, # Untuk memakai idf
174         norm="l2", # Normalisasi
175         smooth_idf=True, # Untuk mencegah divide-by-zero errors
176     )
177
178     tfidf_matrix = vectorizer.fit_transform(text_content)
179     words = vectorizer.get_feature_names()
180     idf_vector = vectorizer.idf_
181
  
```

Gambar 4.13: Potongan kode *TF IDF*

- Struktur file *PageRank*



Gambar 4.14: Struktur file *PageRank*

- Potongan kode *class* dan *fungsi* pada pengimplementasian *PageRank*

```

152     db_connection = self.db.connect()
153     N = self.db.count_rows(db_connection, "page_information")
154     initial_pr = 1 / N
155     self.save_initial_pagerank(db_connection, initial_pr)
156     self.db.close_connection(db_connection)
157
158     for iteration in range(self.max_iterations):
159         pr_change_sum = 0
160
161         db_connection = self.db.connect()
162         pages = self.get_all_crawled_pages(db_connection)
163
164         for page_row in pages:
165             db_connection.ping()
166
167             page_id = page_row["id_page"]
168             page_url = page_row["url"]
169             current_pagerank = self.get_one_pagerank(db_connection, page_id)
170
171             new_pagerank = 0
172             backlink_ids = set()
173             db_cursor2 = db_connection.cursor(pymysql.cursors.DictCursor)
174
175             db_cursor2.execute(
176                 "SELECT `page_id` FROM `page_linking` WHERE `outgoing_link` = %s",
177                 (page_url),
178             )
179             for page_linking_row in db_cursor2.fetchall():
180                 backlink_ids.add(page_linking_row["page_id"])
181

```

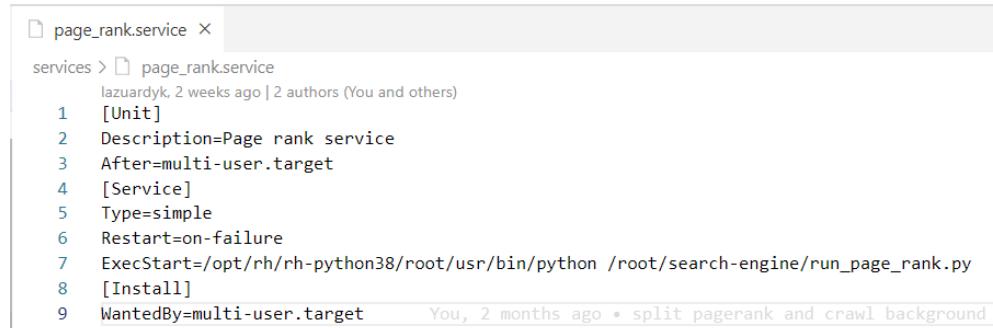
Gambar 4.15: Potongan kode *PageRank*

4. *Sprint-5* (13 Agustus 2022 - 26 Agustus 2022)

Tabel 4.5: *Sprint 5 Backlog*

No	Story	Task	Status
1	Konfigurasi program <i>PageRank</i> dan <i>TF IDF</i>	Buat <i>background service</i> <i>script</i> untuk <i>PageRank</i>	<i>Completed</i>
2	sebagai <i>background service</i> di sistem operasi	Buat <i>background service</i> <i>script</i> untuk <i>TF IDF</i>	<i>Completed</i>
3		Konfigurasi <i>background service</i> <i>TF IDF</i> dan <i>PageRank</i> di <i>server</i>	<i>Completed</i>
4	Perancangan dan implementasi <i>REST API</i>	Merancang <i>web service</i> untuk mengambil data halaman <i>crawling (extraction)</i>	<i>Completed</i>

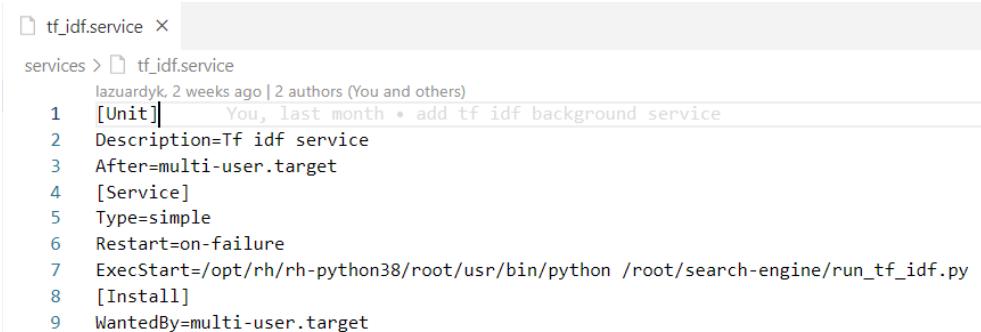
- *Background service script* untuk *PageRank*



```
page_rank.service
services > page_rank.service
lazuardyk, 2 weeks ago | 2 authors (You and others)
1 [Unit]
2 Description=Page rank service
3 After=multi-user.target
4 [Service]
5 Type=simple
6 Restart=on-failure
7 ExecStart=/opt/rh/rh-python38/root/usr/bin/python /root/search-engine/run_page_rank.py
8 [Install]
9 WantedBy=multi-user.target
```

Gambar 4.16: *Background service script PageRank*

- *Background service script* untuk *TF IDF*



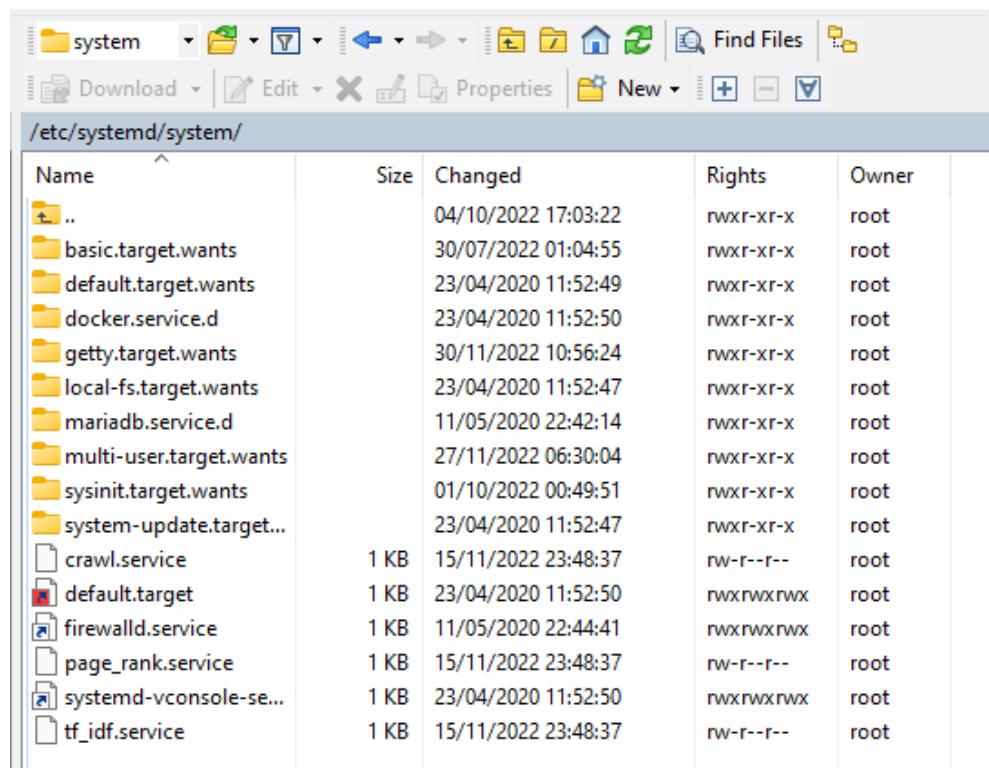
```

tf_idf.service
services > tf_idf.service
lazuardyk, 2 weeks ago | 2 authors (You and others)
1 [Unit]           You, last month • add tf idf background service
2 Description=Tf idf service
3 After=multi-user.target
4 [Service]
5 Type=simple
6 Restart=on-failure
7 ExecStart=/opt/rh/rh-python38/root/usr/bin/python /root/search-engine/run_tf_idf.py
8 [Install]
9 WantedBy=multi-user.target

```

Gambar 4.17: *Background service script TF IDF*

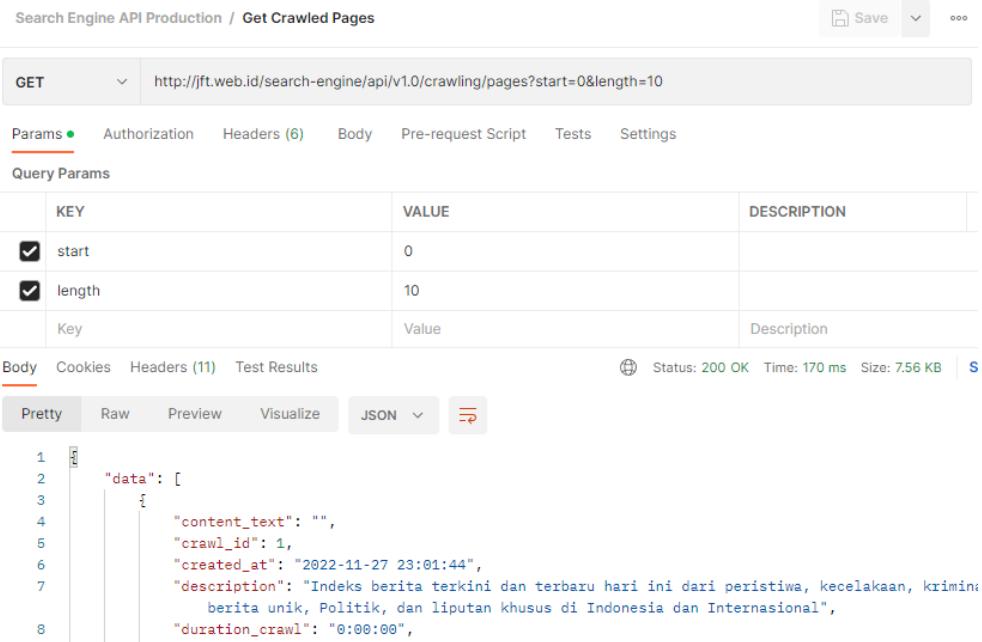
- Konfigurasi *background service* di *server* dengan menggunakan *systemd*



Name	Size	Changed	Rights	Owner
..		04/10/2022 17:03:22	rwxr-xr-x	root
basic.target.wants		30/07/2022 01:04:55	rwxr-xr-x	root
default.target.wants		23/04/2020 11:52:49	rwxr-xr-x	root
docker.service.d		23/04/2020 11:52:50	rwxr-xr-x	root
getty.target.wants		30/11/2022 10:56:24	rwxr-xr-x	root
local-fs.target.wants		23/04/2020 11:52:47	rwxr-xr-x	root
mariadb.service.d		11/05/2020 22:42:14	rwxr-xr-x	root
multi-user.target.wants		27/11/2022 06:30:04	rwxr-xr-x	root
sysinit.target.wants		01/10/2022 00:49:51	rwxr-xr-x	root
system-update.target...		23/04/2020 11:52:47	rwxr-xr-x	root
crawl.service	1 KB	15/11/2022 23:48:37	rw-r--r--	root
default.target	1 KB	23/04/2020 11:52:50	rwxrwxrwx	root
firewalld.service	1 KB	11/05/2020 22:44:41	rwxrwxrwx	root
page_rank.service	1 KB	15/11/2022 23:48:37	rw-r--r--	root
systemd-vconsole-se...	1 KB	23/04/2020 11:52:50	rwxrwxrwx	root
tf_idf.service	1 KB	15/11/2022 23:48:37	rw-r--r--	root

Gambar 4.18: Konfigurasi *background service* di *server*

- REST API untuk mendapatkan data halaman *crawling (extraction)*



The screenshot shows a Postman request for 'Search Engine API Production / Get Crawled Pages'. The method is 'GET' and the URL is 'http://jft.web.id/search-engine/api/v1.0/crawling/pages?start=0&length=10'. The 'Params' tab is selected, showing 'start' with value '0' and 'length' with value '10'. The 'Body' tab shows a JSON response with the following data:

```

1
2   "data": [
3     {
4       "content_text": "",
5       "crawl_id": 1,
6       "created_at": "2022-11-27 23:01:44",
7       "description": "Indeks berita terkini dan terbaru hari ini dari peristiwa, kecelakaan, kriminal, berita unik, Politik, dan liputan khusus di Indonesia dan Internasional",
8       "duration_crawl": "0:00:00",
9     }
10  ]

```

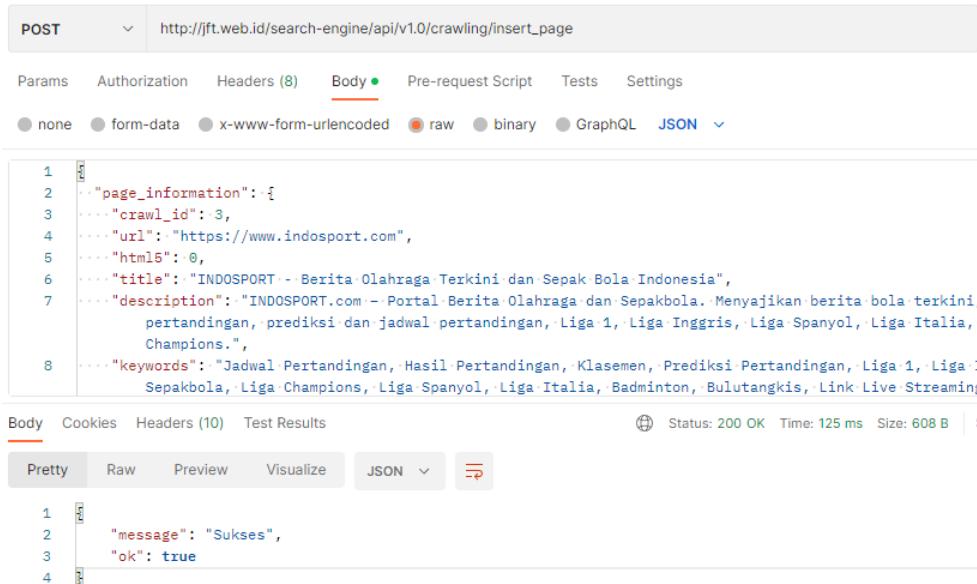
Gambar 4.19: REST API *crawling extraction*

5. Sprint-6 (27 Agustus 2022 - 16 September 2022)

Tabel 4.6: Sprint 6 Backlog

No	Story	Task	Status
1	Perancangan dan implementasi REST API	Merancang <i>web service</i> untuk menambah data halaman <i>crawling (insertion)</i>	Completed
2		Merancang <i>web service</i> untuk mendapatkan skor <i>TF IDF</i>	Completed
3		Merancang <i>web service</i> untuk mendapatkan skor <i>PageRank</i>	Completed

- REST API untuk memasukkan data halaman *crawling* (*insertion*)



The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://jft.web.id/search-engine/api/v1.0/crawling/insert_page
- Body:** JSON (selected)
- Body Content:**

```

1
2   "page_information": {
3     "crawl_id": 3,
4     "url": "https://www.indosport.com",
5     "html5": 0,
6     "title": "INDOSPORT - Berita Olahraga Terkini dan Sepak Bola Indonesia",
7     "description": "INDOSPORT.com - Portal Berita Olahraga dan Sepakbola. Menyajikan berita bola terkini, pertandingan, prediksi dan jadwal pertandingan, Liga 1, Liga Inggris, Liga Spanyol, Liga Italia, Liga Champions.",
8     "keywords": "Jadwal Pertandingan, Hasil Pertandingan, Klasemen, Prediksi Pertandingan, Liga 1, Liga Inggris, Sepakbola, Liga Champions, Liga Spanyol, Liga Italia, Badminton, Bulutangkis, Link Live Streaming"

```
- Response Status:** 200 OK
- Response Time:** 125 ms
- Response Size:** 608 B
- Response Body (Pretty JSON):**

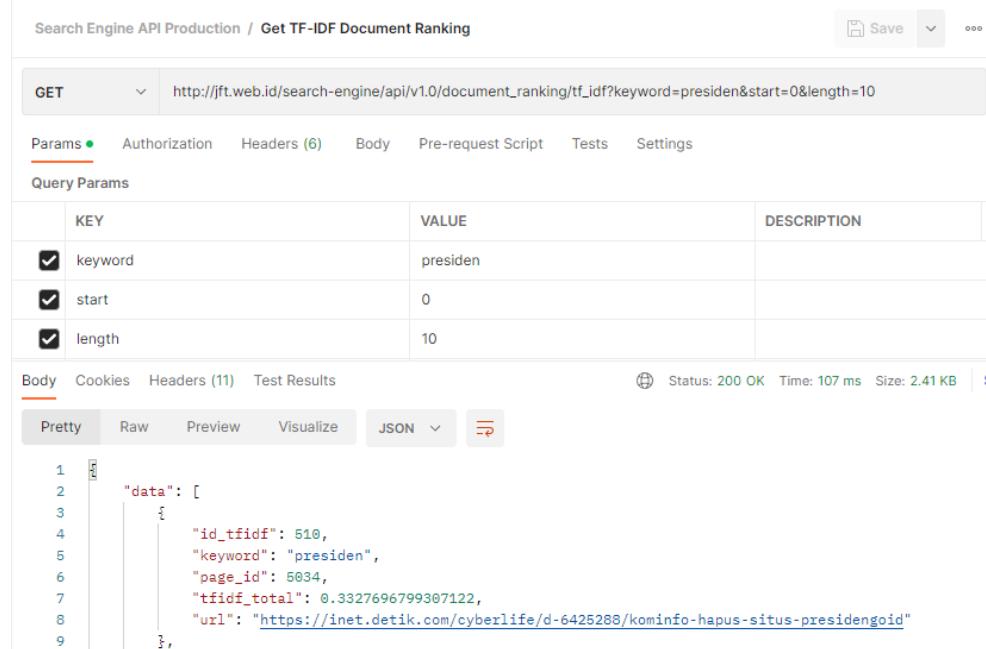
```

1
2   "message": "Sukses",
3   "ok": true
4

```

Gambar 4.20: REST API *crawling insertion*

- REST API untuk mendapatkan ranking *TF IDF*



The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** http://jft.web.id/search-engine/api/v1.0/document_ranking/tf_idf?keyword=presiden&start=0&length=10
- Params:** keyword: presiden, start: 0, length: 10
- Response Status:** 200 OK
- Response Time:** 107 ms
- Response Size:** 2.41 KB
- Response Body (Pretty JSON):**

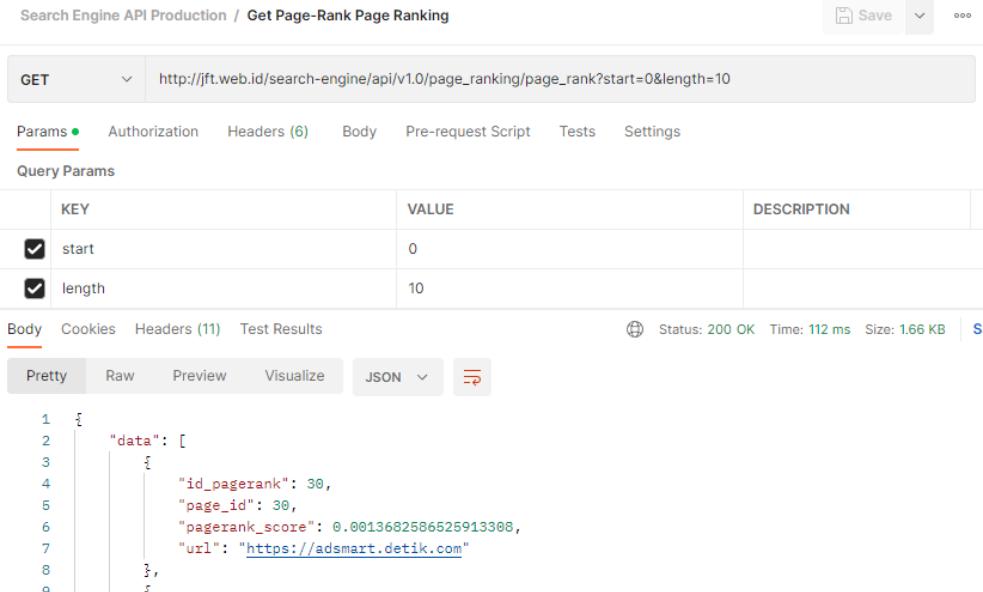
```

1
2   "data": [
3     {
4       "id_tfidf": 510,
5       "keyword": "presiden",
6       "page_id": 5034,
7       "tfidf_total": 0.3327696799307122,
8       "url": "https://inet.detik.com/cyberlife/d-6425288/kominfo-hapus-situs-presidengoid"
9     }

```

Gambar 4.21: REST API ranking *TF IDF*

- REST API untuk mendapatkan ranking PageRank



Search Engine API Production / Get Page-Rank Page Ranking

GET http://jft.web.id/search-engine/api/v1.0/page_ranking/page_rank?start=0&length=10

Params (6) Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> start	0	
<input checked="" type="checkbox"/> length	10	

Body Cookies Headers (11) Test Results Status: 200 OK Time: 112 ms Size: 1.66 KB

Pretty Raw Preview Visualize JSON

```

1  {
2    "data": [
3      {
4        "id_pageRank": 30,
5        "page_id": 30,
6        "pagerank_score": 0.0013682586525913308,
7        "url": "https://adsmart.detik.com"
8      }
9    ]

```

Gambar 4.22: REST API ranking PageRank

6. Sprint-7 (17 September 2022 - 30 September 2022)

Tabel 4.7: Sprint 7 Backlog

No	Story	Task	Status
1	Perancangan dan implementasi REST API	Merancang <i>web service</i> untuk dapat menjalankan <i>crawling</i> dengan durasi tertentu dalam satuan detik	Completed
2		<i>Testing keyword</i> menggunakan spasi pada <i>URL web service</i>	Completed
3		Catat ukuran halaman yang di- <i>crawl</i> pada <i>database</i>	Completed

- REST API untuk menjalankan *crawler* dengan batasan detik

Search Engine API Production / Run Crawling Save ...

GET ▼ <http://jft.web.id/search-engine/api/v1.0/crawling/start?duration=10>

Params ● Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> duration	10	

Body Cookies Headers (10) Test Results >Status: 200 OK Time: 104 ms Size: 610 B ☰

Pretty Raw Preview Visualize JSON ☰

```
1
2   "message": "Sukses",
3   "ok": true
4
```

Gambar 4.23: REST API untuk menjalankan crawler

- Percobaan *keyword* dengan spasi pada *URL REST API*, hasilnya berhasil

Search Engine API Production / Get TF-IDF Document Ranking

Save   

GET  http://jft.web.id/search-engine/api/v1.0/document_ranking/tf_idf?keyword=resesi%202023&start=0&length=10

Params  Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> keyword	resesi 2023	
<input checked="" type="checkbox"/> start	0	
<input checked="" type="checkbox"/> length	10	

Body Cookies Headers (11) Test Results  Status: 200 OK Time: 87 ms Size: 2.41 KB 

Pretty Raw Preview Visualize  

```
1  "data": [
2   {
3     "id_tfidf": 779,
4     "keyword": "resesi 2023",
5     "page_id": 4901,
6     "tfidf_total": 0.6187756986195736,
7     "url": "https://www.detik.com/edu/detikpedia/d-6417781/
8       apa-itu-resesi-begini-dampak-dan-kiat-menghadapinya"
9   }
]
```

Gambar 4.24: Percobaan *keyword* memakai spasi pada *URL*

- Ukuran halaman dalam satuan *bytes* pada *database*

description	keywords	content_text	hot_url	size_bytes	model_crawl
Rangkuman	Berita	Statistik Gila			
Berita	Terpopuler	Lionel			
Terpopuler dan	Hari Ini	Messi, Lewati	0	126889	BFS crawling
Terbaru Hari Ini	Detikcom, d...	Haaland dan berita terbar...	Lew...		
-	-	detikNetwork adalah bagian dari CT Corp. Satu akun...	0	3196	BFS crawling
Tonton siaran TV online dari Trans TV, live stream...	TransTV, My Trip My Adventure, Insert, Brownis, Dr...	MENU detikcom NEW NEW News Finance Teknologi Enter...	0	87595	BFS crawling

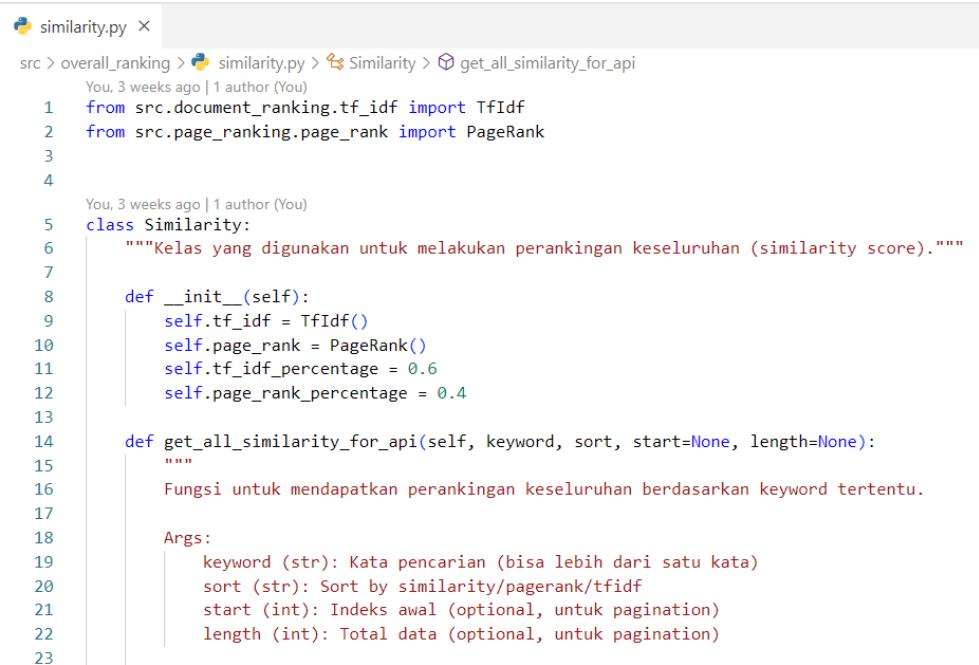
Gambar 4.25: Ukuran halaman di *database*

7. Sprint-8 (1 Oktober 2022 - 7 Oktober 2022)

Tabel 4.8: *Sprint 8 Backlog*

No	Story	Task	Status
1	Integrasi <i>PageRank</i> dan <i>TF IDF</i> untuk menghasilkan <i>similarity score</i>	Buat spesifikasi struktur <i>class</i> dan fungsi untuk <i>similarity score</i>	<i>Completed</i>
2		Implementasi <i>similarity score</i> dengan mengintegrasikan skor <i>TF IDF</i> dan <i>PageRank</i>	<i>Completed</i>
3		Buat <i>web service</i> untuk mendapatkan <i>similarity score</i>	<i>Completed</i>

- Potongan kode pengimplementasian *similarity score*



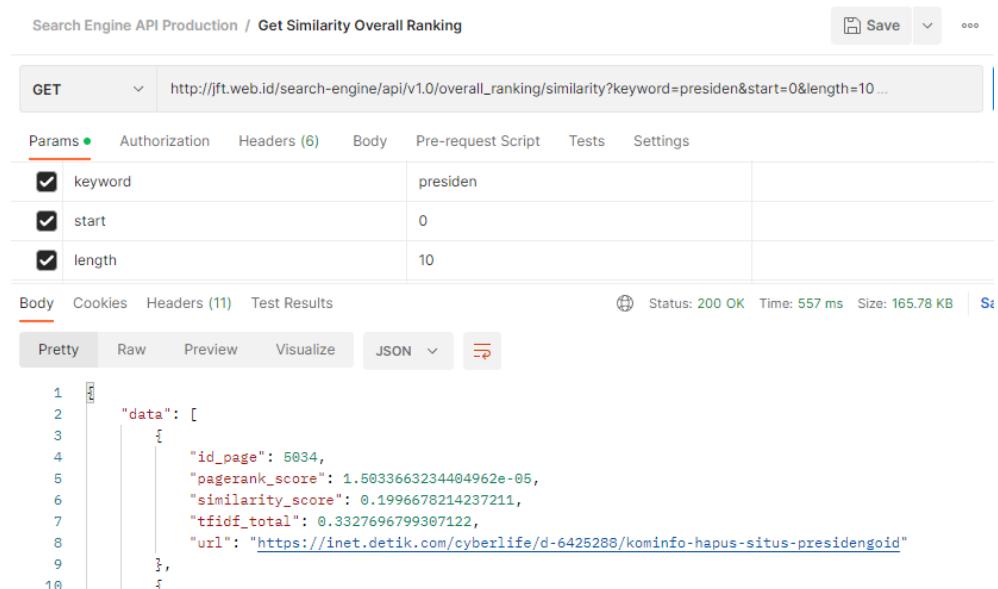
```

similarity.py < src > overall_ranking > similarity.py > Similarity > get_all_similarity_for_api
You, 3 weeks ago | 1 author (You)
1  from src.document_ranking.tf_idf import TfIdf
2  from src.page_ranking.page_rank import PageRank
3
4
5  You, 3 weeks ago | 1 author (You)
6  class Similarity:
7      """Kelas yang digunakan untuk melakukan perankingan keseluruhan (similarity score)."""
8
9      def __init__(self):
10          self.tf_idf = TfIdf()
11          self.page_rank = PageRank()
12          self.tf_idf_percentage = 0.6
13          self.page_rank_percentage = 0.4
14
15      def get_all_similarity_for_api(self, keyword, sort, start=None, length=None):
16          """
17          Fungsi untuk mendapatkan perankingan keseluruhan berdasarkan keyword tertentu.
18
19          Args:
20              keyword (str): Kata pencarian (bisa lebih dari satu kata)
21              sort (str): Sort by similarity/pagerank/tfidf
22              start (int): Indeks awal (optional, untuk pagination)
23              length (int): Total data (optional, untuk pagination)

```

Gambar 4.26: Potongan kode *similarity score*

- *REST API* untuk mendapatkan *similarity score*



Params		Authorization	Headers (6)	Body	Pre-request Script	Tests	Settings
<input checked="" type="checkbox"/>	keyword			presiden			
<input checked="" type="checkbox"/>	start			0			
<input checked="" type="checkbox"/>	length			10			

Body: Status: 200 OK Time: 557 ms Size: 165.78 KB

```

1
2  "data": [
3      {
4          "id_page": 5034,
5          "pagerank_score": 1.5033663234404962e-05,
6          "similarity_score": 0.1996678214237211,
7          "tfidf_total": 0.3327696799307122,
8          "url": "https://inet.detik.com/cyberlife/d-6425288/kominfo-hapus-situs-presidengoid"
9      },
10     {

```

Gambar 4.27: *REST API similarity score*

8. *Sprint-9* (8 Oktober 2022 - 21 Oktober 2022)

Tabel 4.9: Sprint 9 Backlog

No	Story	Task	Status
1	Buat program berbasis <i>console</i> dan tampilan web sederhana untuk melakukan pencarian	Revisi pada <i>web service similarity score</i> untuk menambahkan opsi agar dapat memilih urutan berdasarkan skor <i>similarity</i> , <i>TF IDF</i> , atau <i>PageRank</i>	<i>Completed</i>
2		Buat program <i>console</i> yang menerima <i>keyword</i> dan menampilkan hasil <i>web</i> yang relevan	<i>Completed</i>
3		Buat tampilan web sederhana untuk menerima <i>keyword</i> dan menampilkan hasil <i>web</i> yang relevan beserta dengan skornya	<i>Completed</i>

- REST API *similarity score* dengan parameter *sort* untuk memilih urutan.

Urutan yang bisa dipilih adalah *similarity*, *PageRank*, atau *TF IDF*

Search Engine API Production / Get Similarity Overall Ranking

GET http://jft.web.id/search-engine/api/v1.0/overall_ranking/similarity?keyword=presiden&start=0&length=10&sort=page...

Params Authorization Headers (6) Body Pre-request Script Tests Settings

KEY	VALUE	DESCRIPTION
keyword	presiden	
start	0	
length	10	
sort	pagerank	

Body Cookies Headers (11) Test Results Status: 200 OK Time: 632 ms Size: 165.79 KB

Pretty Raw Preview Visualize JSON

```

1   "data": [
2     {
3       "id_page": 352,
4       "pagerank_score": 0.00010868337258857051,
5       "similarity_score": 0.07817657370573881,
6       "tfidf_total": 0.13022183392783898,
7       "url": "https://www.detik.com/tag/resesi"
8     },
9   ],

```

Gambar 4.28: REST API *similarity score* dengan urutan berdasarkan *PageRank*

- Program *search engine* berbasis *console*

```

[root@jft search-engine]# /opt/rh/rh-python38/root/usr/bin/python run_search_engine_console.py
Input keyword pencarian: resesi 2023
Hasil pencarian:

1. Apa Itu Resesi? Begini Dampak dan Kiat Menghadapinya
https://www.detik.com/edu/detikpedia/d-6417781/apa-itu-resesi-begini-dampak-dan-kiat-menghadapinya
Trisna Wulandari - detikEdu Senin,21 Nov 2022 15:30 WIB 0 komentar URL telah disalin Dampak resesi

2. Podcast: Anti Panic Panic Club Ramalan Ekonomi Gelap 2023
https://finance.detik.com/berita-ekonomi-bisnis/d-6361031/podcast-anti-panic-panic-club-ramalan-ekonomi-gelap-2023
Tolak Miskin Podcast: Anti Panic Panic Club Ramalan Ekonomi Gelap 2023 Eduardo Simorangkir - detikFi

3. Lampu Kuning Bagi Kaum Pekerja Jelang Masa Resesi
https://finance.detik.com/berita-ekonomi-bisnis/d-6365111/lampu-kuning-bagi-kaum-pekerja-jelang-masa-resesi
Lampu Kuning Bagi Kaum Pekerja Jelang Masa Resesi Edward. F. Kusuma - detikFinance Senin,24 Okt 2022

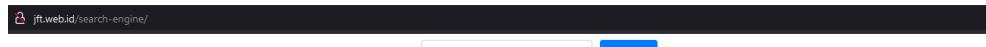
4. Zulhas Geber Pasar Dalam Negeri-Ekspor di Tengah Ancaman Resesi Global
https://finance.detik.com/berita-ekonomi-bisnis/d-6420456/zulhas-geber-pasar-dalam-negeri-ekspor-di-tengah-ancaman-resesi-global
Zulhas Geber Pasar Dalam Negeri-Ekspor di Tengah Ancaman Resesi Global Mega Putra Ratya - detikFinan

5. Berita dan Informasi Resesi Terkini dan Terbaru Hari ini - detikcom
https://www.detik.com/tag/resesi/?sortby=time&page=3
detikFinance Selasa,15 Nov 2022 08:30 WIB Ekonomi Global Dihantui Resesi,Investasi Apa yang Masih Bi

```

Gambar 4.29: Program *search engine* berbasis *console*

- Tampilan web *search engine* yang menampilkan *URL* yang relevan beserta dengan skornya



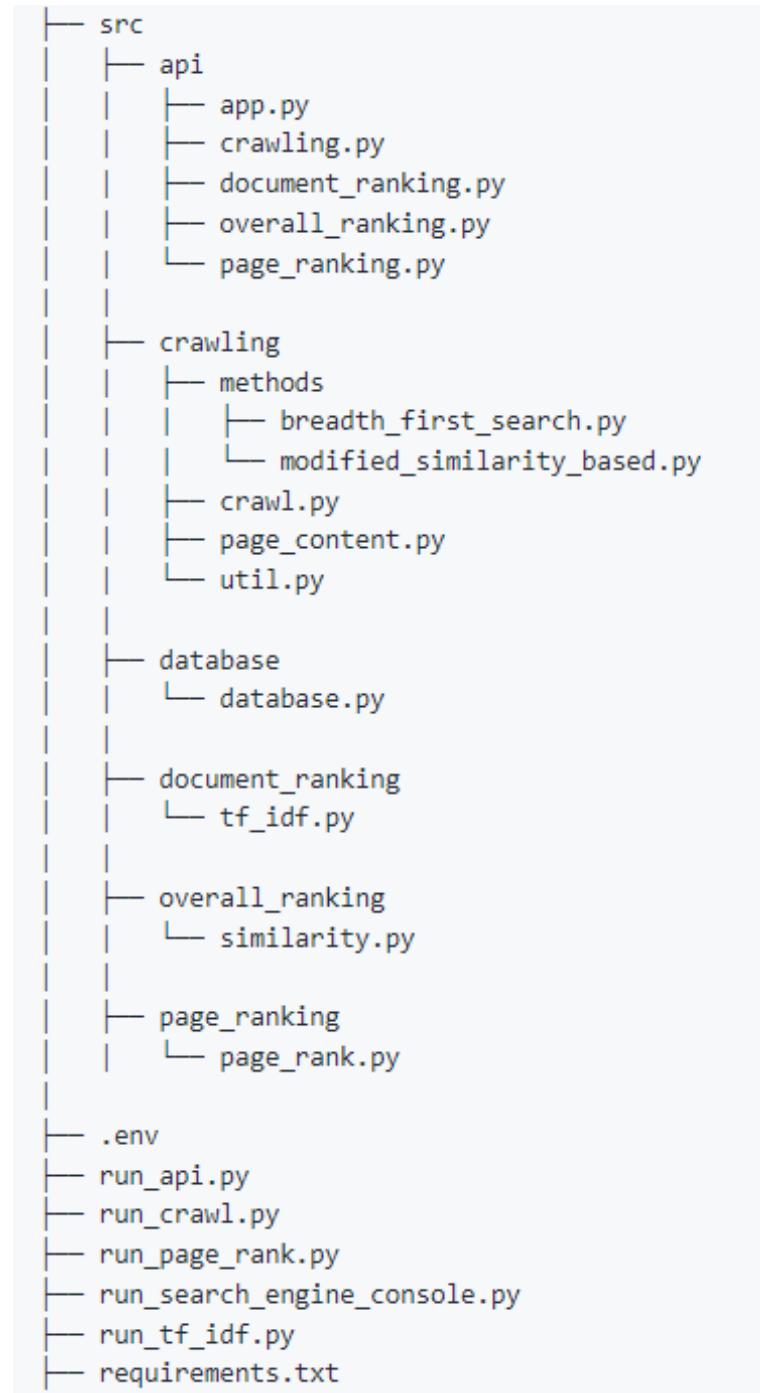
The screenshot shows a search engine interface with the URL ift.web.id/search-engine/ in the address bar. The search query 'resesi 2023' is entered in the search bar, and a 'Submit' button is visible. Below the search bar is a table with the following columns: ID, URL, Page Rank Score, Cosine (TF IDF) Score, and Overall Similarity. The table lists 14 search results, each with a unique ID, a URL, and its corresponding scores.

ID	URL	Page Rank Score	Cosine (TF IDF) Score	Overall Similarity
4901	https://www.detik.com/edu/detikpedia/d-6417781/apa-itu-resesi-begini-dampak-dan-kiat-menghadapinya	0.000016677494428104144	0.6187756986195736	0.6187923761140017
4928	https://finance.detik.com/berita-ekonomi-bisnis/d-6361031/podcast-anti-panic-panic-club-ramalan-ekonomi-gelap-2023	0.00001589827949339418	0.5526985020261157	0.5527144003056091
9878	https://finance.detik.com/berita-ekonomi-bisnis/d-6365111/lampu-kuning-bagi-kaum-pekerja-jelang-masa-resesi	0.000014395936204761532	0.520381658029325	0.5203960539655298
4900	https://finance.detik.com/berita-ekonomi-bisnis/d-6420456/zulhas-geber-pasar-dalam-negeri-ekspor-di-tengah-ancaman-resesi-global	0.000015213560151047635	0.47297557335904283	0.4729907869191939
4906	https://www.detik.com/tag/resesi/?sortby=time&page=3	0.000015499821873190325	0.4375200016278873	0.4375355014497605
4645	https://finance.detik.com/berita-ekonomi-bisnis/d-6384320/3-jurusan-hadapi-resesi-ekonomi-untuk-umkm-ala-sandiaga-uno	0.000014436391260544758	0.41748501353824874	0.41749944992950927
4899	https://finance.detik.com/berita-ekonomi-bisnis/d-6420569/dunia-hadapi-ancaman-baru-bernama-resflasi-ri-aman	0.000015552755722746297	0.39527926054788926	0.395294813303612
4895	https://www.detik.com/tag/foto/resesi	0.000015754902231099942	0.38824921055290573	0.38826496545513683
4147	https://finance.detik.com/infografis/d-6358197/negara-negara-rantau-resesi-versi-sri-mulyani	0.000014699434798784021	0.3809822789778948	0.38099697841269353
807	https://health.detik.com/berita-detikhealth/d-6428985/korsel-dibayangi-resesi-seks-begini-pengakuan-warga-yang-ogah-punya-ank	0.000023681802597243547	0.3679237635879884	0.36794744539058566
4908	https://www.detik.com/tag/resesi/?sortby=time&page=5	0.00001549982125207239	0.3491250448996161	0.3491405447208682
2061	https://www.cnbcindonesia.com/news/20221127200319-16-391611/3-negara-asia-dihantui-resesi-seks-populasi-manusia-terancam	0.000030958044160907635	0.34001341900348975	0.34004437704765067

Gambar 4.30: Tampilan sederhana web *search engine*

B. Pembahasan Arsitektur

1. Struktur Direktori dan *File*



Gambar 4.31: Struktur direktori dan *file*

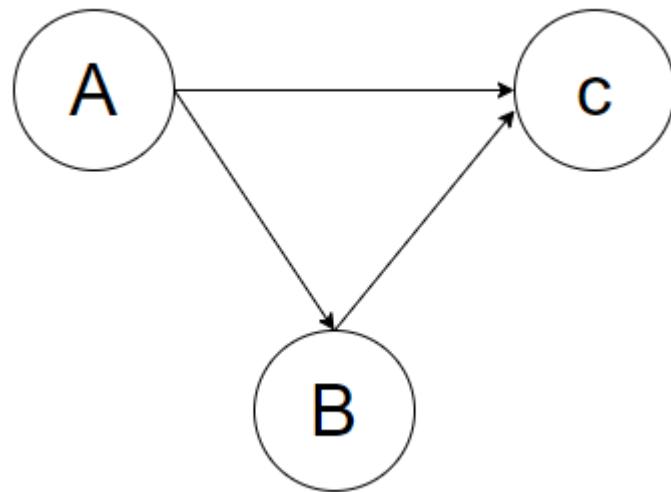
Spesifikasi struktur direktori dan *file search engine* dibagi menjadi beberapa *package* atau folder sesuai dengan komponennya masing-masing. Pada direktori terluar atau *root project* dikhususkan pada *file* konfigurasi *search engine* dan *script* yang nantinya dapat dijalankan secara langsung. Sedangkan algoritma atau inti kode dari tiap komponennya terdapat di dalam direktori *src*.

Di dalam folder *src*, kode *search engine* dibagi lagi menjadi beberapa direktori antara lain direktori *api* yang berisi kode yang berhubungan dengan *web service*, direktori *crawling* yang berisi kode pemrosesan *crawling*, direktori *database* yang berisi kode untuk melakukan koneksi ke *database*, direktori *document_ranking* yang berisi kode algoritma relevansi pencarian berdasarkan isi konten atau dokumen *website*, direktori *page_ranking* yang berisi kode perankingan berdasarkan kepopuleran suatu website tanpa memedulikan kontennya, dan direktori *overall_ranking* yang berisi kode untuk mengombinasikan semua perankingan yang ada dan menghasilkan hasil pencarian yang relevan.

2. *Overall Similarity Score*

Overall similarity score merupakan nilai gabungan dari *document ranking* dan *page ranking* yang digunakan sebagai nilai akhir untuk hasil pencarian *search engine*. Nilai ini dihitung menggunakan metode *cosine similarity* lalu dijumlahkan dengan nilai *PageRank*. Nilai terbesar akan menempati posisi pertama sebagai yang paling relevan, dan sebaliknya.

Cosine similarity merupakan nilai kemiripan yang dihitung berdasarkan bobot *TF IDF* dokumen dan *keyword*. Nilai kemiripan ini didapatkan dari *dot product* antara vektor *keyword* dan vektor dokumen yang merupakan jumlah hasil kali dari kedua vektor tersebut. Setelah itu, nilai *cosine* dijumlahkan dengan nilai *PageRank* dan menghasilkan nilai *overall similarity score*.



Gambar 4.32: Contoh halaman dengan arah *link*

Sebagai contoh, terdapat halaman A, B, dan C seperti pada gambar 4.32. Halaman A memiliki link yang mengarah ke B dan C, halaman B memiliki link yang mengarah ke C, dan halaman C tidak memiliki link ke arah lain. Selain itu, tiap-tiap halaman memiliki isi konten teks sebagai berikut:

Tabel 4.10: Contoh halaman dengan isi konten

Halaman	Konten Teks
A	Kucing merupakan hewan mamalia
B	Sapi adalah hewan ternak
C	Hewan mamalia adalah hewan yang menyusui

Dari data halaman dengan arah *link* dan isi konten di atas, akan dilakukan pencarian terhadap query atau keyword "**mamalia adalah**".

Perhitungan *PageRank* pada tiap-tiap halaman berdasarkan gambar 4.32 adalah sebagai berikut:

$$PR(A) = ((1 - 0.85) / 3) = 0.05$$

$$PR(B) = ((1 - 0.85) / 3) + 0.85 * (0.05 / 2) = 0.07125$$

$$PR(C) = ((1 - 0.85) / 3) + 0.85 * (0.05 / 2) + 0.85 * (0.07125 / 1) = 0.13181$$

Perhitungan *TF IDF* pada tiap-tiap halaman dan pada *query* adalah sebagai berikut:

$$TF-IDF (\text{kucing}, A): 1/4 \times \log(3/1) = 0.27465$$

$$TF-IDF (\text{merupakan}, A): 1/4 \times \log(3/1) = 0.27465$$

$$TF-IDF (\text{hewan}, A): 1/4 \times \log(3/3) = 0.0$$

$$TF-IDF (\text{mamalia}, A): 1/4 \times \log(3/2) = 0.10136$$

$$TF-IDF (\text{sapi}, B): 1/4 \times \log(3/1) = 0.27465$$

$$TF-IDF (\text{adalah}, B): 1/4 \times \log(3/2) = 0.10136$$

$$TF-IDF (\text{hewan}, B): 1/4 \times \log(3/3) = 0.0$$

$$TF-IDF (\text{ternak}, B): 1/4 \times \log(3/1) = 0.27465$$

$$TF-IDF (\text{hewan}, C): 2/5 \times \log(3/3) = 0.0$$

$$TF-IDF (\text{mamalia}, C): 1/5 \times \log(3/2) = 0.08109$$

$$TF-IDF (\text{adalah}, C): 1/5 \times \log(3/2) = 0.08109$$

$$TF-IDF (\text{menyusui}, C): 1/5 \times \log(3/1) = 0.21972$$

$$TF-IDF (\text{mamalia}, Q): 1/2 \times \log(3/2) = 0.20273$$

$$TF-IDF (\text{adalah}, Q): 1/2 \times \log(3/2) = 0.20273$$

Selanjutnya, perhitungan *cosine similarity* adalah sebagai berikut:

$$\text{Cosine (query, halaman A)} = \text{Vektor A} \times \text{Vektor Query} / \|\text{Query}\| \times \|\text{Document A}\|$$

$$\begin{aligned} \text{Cosine (query, halaman A)} &= (0.20273 * 0.10136) + 0.0 / (0.27465 + 0.27465 + 0.0 \\ &+ 0.10136) * (0.20273 + 0.20273) = 0.07789 \end{aligned}$$

$$\begin{aligned} \text{Cosine (query, halaman B)} &= 0.0 + (0.20273 * 0.10136) / (0.27465 + 0.10136 + 0.0 \\ &+ 0.27465) * (0.20273 + 0.20273) = 0.07789 \end{aligned}$$

$$\begin{aligned} \text{Cosine (query, halaman C)} &= (0.20273 * 0.08109) + (0.20273 * 0.08109) / (0.0 \\ &+ 0.08109 + 0.08109 + 0.21972) * (0.20273 + 0.20273) = 0.21233 \end{aligned}$$

Setelah itu, overall similarity score dapat didapatkan dengan menjumlahkan nilai cosine similarity dengan nilai PageRank:

$$\text{Overall score halaman A} = 0.07789 + 0.05 = \mathbf{0.12789}$$

$$\text{Overall score halaman B} = 0.07789 + 0.07125 = \mathbf{0.14914}$$

$$\text{Overall score halaman C} = 0.21233 + 0.13181 = \mathbf{0.34414}$$

Nilai yang memiliki *overall similarity score* yang paling tinggi merupakan halaman yang paling relevan. Dengan demikian, urutan halaman yang paling relevan dengan query "**mamalia adalah**" adalah halaman C, halaman B, dan halaman A seperti pada tabel 4.11.

Tabel 4.11: Hasil relevansi dari *query* "mamalia adalah"

Halaman	Konten Teks
C	Hewan mamalia adalah hewan yang menyusui
B	Sapi adalah hewan ternak
A	Kucing merupakan hewan mamalia

3. *Technology Stack*

Technology stack merupakan daftar teknologi yang dipakai dalam suatu sistem aplikasi. Kode utama program *search engine* ini dibangun dengan bahasa pemrograman *Python 3*, dan sistem basis data yang dipakai adalah *MySQL*.



```
12 lines (12 sloc) | 109 Bytes
1 beautifulsoup4
2 requests
3 pandas
4 pymysql
5 python-dotenv
6 psutil
7 sklearn
8 numpy
9 pdoc3
10 matplotlib
11 flask
12 cryptography
```

Gambar 4.33: *Library Python*

Gambar 4.33 merupakan daftar library *Python* yang digunakan dalam sistem ini. Diantaranya yaitu: *beautifulsoup4*, *requests*, *pandas*, *pymysql*, *python-dotenv*, *psutil*, *sklearn*, *numpy*, *pdoc3*, *matplotlib*, *flask*, *cryptography*.

4. Cara Kerja

Cara kerja arsitektur search engine ini pada suatu server adalah sebagai berikut.

1. *Crawler* berjalan di *background* dengan menggunakan *script run_crawl.py*
2. Setelah *crawler* telah selesai, perangkingan *PageRank* dan pembobotan *TF IDF* dijalankan di *background* dengan menggunakan *script run_page_rank.py* dan *run_tf_idf.py*
3. Setelah *PageRank* dan *TF IDF* selesai, *web service* dapat dijalankan menggunakan *script run_api.py*
4. Tampilan sudah dapat diakses untuk melakukan pencarian, alternatif pencarian dapat dilakukan melalui *console* dengan *script run_search_engine_console.py*

C. Pengujian Arsitektur

Pengujian arsitektur *search engine* dilakukan menggunakan teknik *functional* dan *non-functional testing*.

1. *Functional Testing*

Functional testing merupakan pengujian untuk memverifikasi bahwa setiap fungsi atau fitur pada aplikasi sudah sesuai. Pengujian ini dilakukan secara langsung bersama dengan *Scrum Master*, dokumentasi saat melakukan *functional testing* terdapat pada **Lampiran B**.

Tabel 4.12: Functional Testing

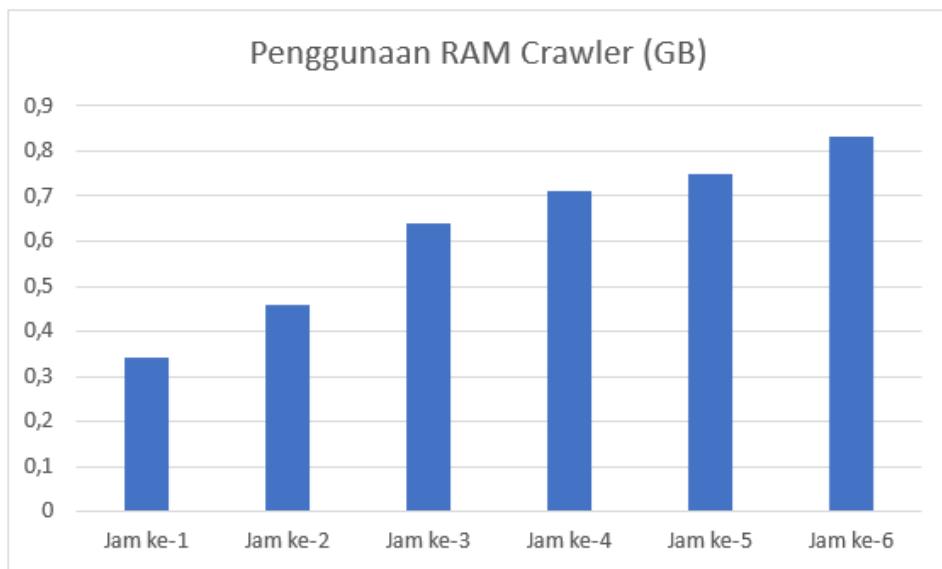
Skenario Pengujian	Yang Diharapkan	Pengamatan	Kesimpulan
Menjalankan program <i>crawler</i> , <i>page rank</i> dan <i>TF IDF</i> di lokal komputer dan melihat hasilnya di <i>database</i>	Data tersimpan dengan baik di <i>database</i>	Data yang dihasilkan di <i>database</i> sudah benar dan sesuai	Diterima
Melakukan <i>start</i> dan <i>stop</i> pada setiap <i>background service</i> di <i>server</i>	<i>Start</i> dan <i>stop</i> pada <i>background service</i> di <i>server</i> dapat bekerja dengan baik	<i>Start</i> dan <i>stop</i> pada <i>background service</i> bekerja dengan baik	Diterima
Menjalankan semua <i>routes</i> yang ada pada <i>REST API</i> dengan <i>Postman</i>	Kode dan isi <i>response</i> tiap <i>endpoint</i> benar dan sesuai fungsinya	Tiap <i>response</i> pada <i>endpoint API</i> sudah benar	Diterima
Melakukan pencarian pada tampilan web dan program <i>console search engine</i>	Mempunyai hasil pencarian yang relevan	Hasil pencarian sudah relevan sesuai <i>keyword</i>	Diterima

2. Non-functional Testing

Non-functional testing pada arsitektur *search engine* ini dilakukan dengan teknik *performance testing*, yaitu sebuah pengujian yang berfokus pada ketahanan dan kestabilan sebuah sistem.

1. Lama waktu dan penggunaan *RAM*

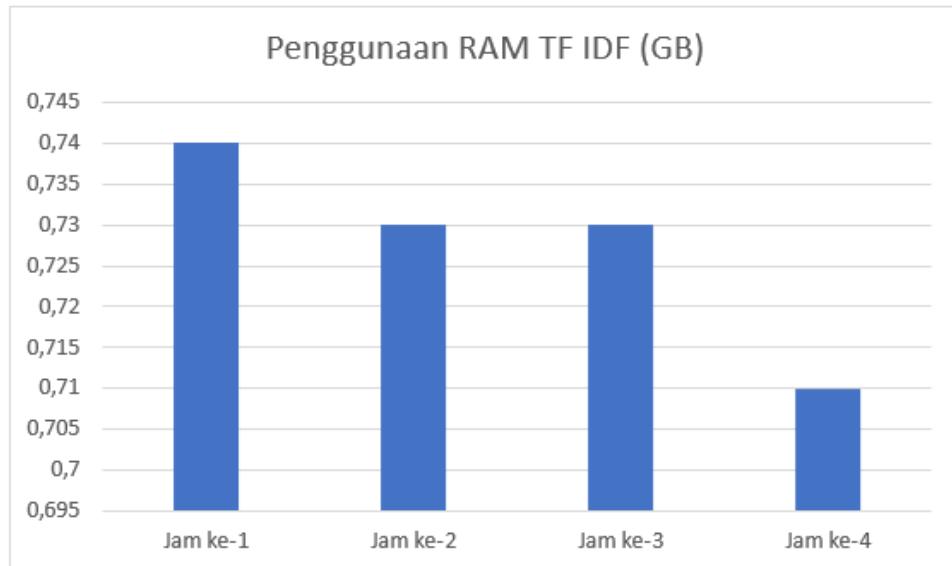
Komponen *Crawler*, *TF IDF*, dan *PageRank* diuji dengan cara menjalankannya langsung di *server* menggunakan *background service* dan mencatat penggunaan memori (*RAM*) beserta lama waktu yang diperlukan dari tiap komponennya.



Gambar 4.34: *Grafik penggunaan RAM pada Crawler*

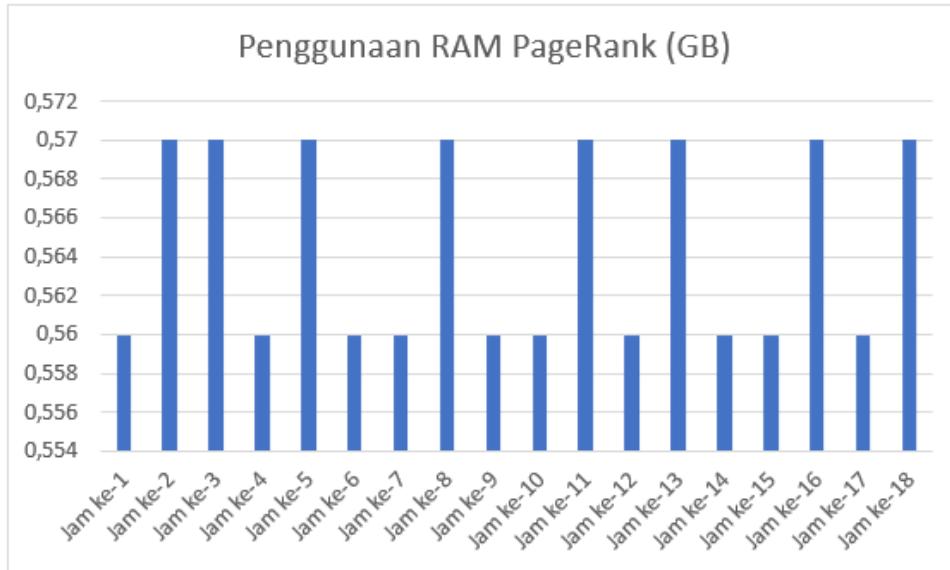
Saat menjalankan *background service crawler* di *server* situs awal yang dipilih adalah detik.com, lalu durasinya diatur pada 6 jam dan dengan jumlah 1 *thread*. Grafik penggunaan *RAM* tiap jam saat *crawler* berjalan dapat dilihat pada gambar 4.34. Berdasarkan grafik, penggunaan *RAM* pada *crawler* bertambah seiring berjalannya waktu, hal ini dikarenakan terdapat antrian halaman yang ingin di-*crawl*. Saat *crawler* memproses setiap halaman dan mengurangi antrian, *crawler* juga menambahkan semua *URL* yang ada pada halaman tersebut ke dalam antrian, sehingga antrian halaman akan lebih banyak bertambah dibandingkan berkurang yang mengakibatkan

bertambahnya konsumsi *RAM*. Hasil halaman yang telah di-*crawl* dari proses *background service* ini berjumlah 10.714 halaman.



Gambar 4.35: Grafik penggunaan *RAM* pada *TF IDF*

Setelah proses *crawler* selesai, *background service TF IDF* dijalankan untuk memproses data halaman yang dihasilkan dari *crawler* tersebut. Grafik penggunaan *RAM* tiap jam saat *TF IDF* berjalan dapat dilihat pada gambar 4.35 di mana penggunaan *RAM*-nya cenderung menurun. *Background service TF IDF* yang memproses 10.714 halaman ini memakan waktu selama 4 jam 23 menit.



Gambar 4.36: Grafik penggunaan *RAM* pada *PageRank*

Background service PageRank juga dijalankan memakai data halaman dari proses *crawler* yang telah dilakukan sebelumnya. Grafik penggunaan *RAM* tiap jam saat *PageRank* berjalan dapat dilihat pada gambar 4.35 di mana penggunaan *RAM*-nya cenderung naik turun tetapi tetap stabil. Proses *PageRank* berhenti atau konvergen pada iterasi ke-8 di mana rata-rata perbedaan nilai *PageRank* di iterasi terakhir bernilai 0.0000636. *Background service PageRank* yang memproses 10.714 halaman ini memakan waktu selama 18 jam 3 menit.

2. Response Time API

Pengujian *REST API* dilakukan dengan membandingkan *response time* tiap *endpoint API* pada saat *background service* berjalan dan pada saat *background service* tidak berjalan. Hasil dari perbandingan ini dapat dilihat pada tabel 4.13.

Tabel 4.13: Perbandingan *response time API*

Route	Response Time	
	Background Service Berjalan	Background Service Tidak Berjalan
/api/v1.0/crawling/crawl	659 ms	302 ms
/api/v1.0/document_ranking/tf_idf	323 ms	304 ms
/api/v1.0/page_ranking/page_rank	534 ms	292 ms
/api/v1.0/overall_ranking/similarity	77 ms	64 ms
/api/v1.0/crawling/pages	92 ms	82 ms
/api/v1.0/crawling/page_information	62 ms	60 ms
/api/v1.0/crawling/start_insert	63 ms	62 ms
/api/v1.0/crawling/insert_page	78 ms	76 ms

Response time API pada saat *background service* berjalan cenderung lebih besar dibandingkan dengan saat tidak berjalan. Perbedaan waktu yang cukup signifikan terjadi pada *route crawling* karena saat pemanggilan tersebut terdapat *thread* lain yang muncul untuk melakukan proses *crawling*.

3. Keandalan *Server*

Keandalan *server* diuji dengan cara melakukan sejumlah *request* yang telah ditentukan ke salah satu *route API*, kode pengujian ini terdapat pada **Lampiran C**.

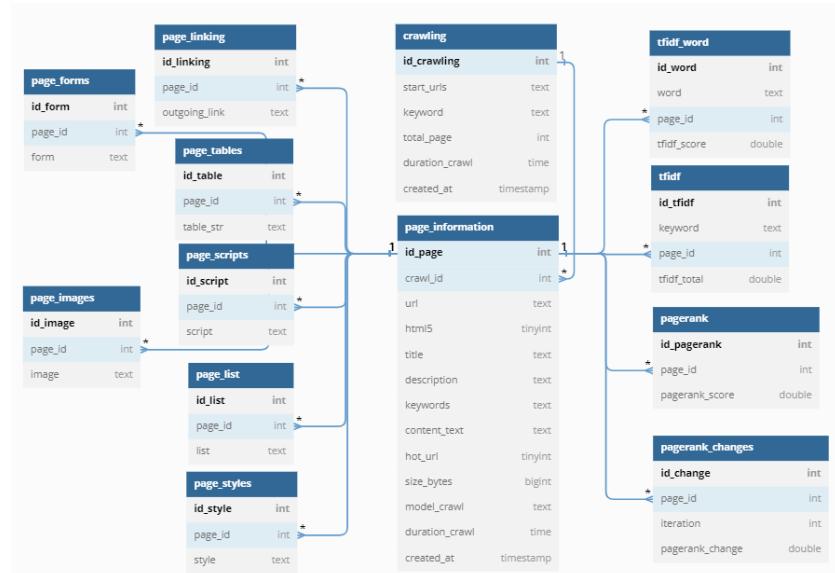
Tabel 4.14: Uji keandalan *server*

Jumlah Request	Rata-rata Response Time
100	581 ms
1000	796 ms
10000	1826 ms

Berdasarkan hasil uji keandalan *server* pada tabel 4.14, dapat disimpulkan bahwa perbedaan *response time* yang signifikan dapat dialami ketika melakukan *request* atau melakukan kunjungan ke *server* selama 10000 kali.

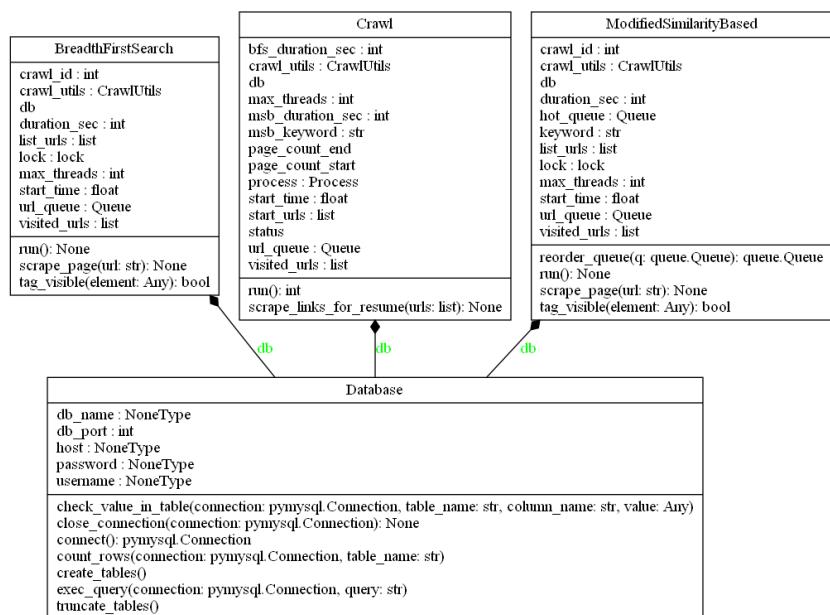
D. Hasil Keseluruhan

1. Desain database sistem



Gambar 4.37: Desain database sistem

2. Class diagram



Gambar 4.38: Class diagram sistem

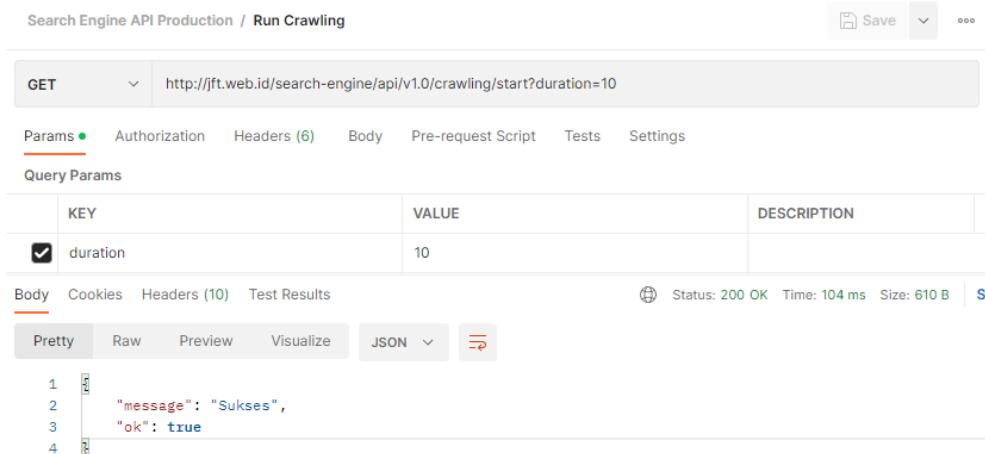
3. Routing table REST API

Tabel 4.15: Routing table REST API

Name	Route	Method	Description	Body	Parameter
Run Crawling	/api/v1.0/crawling/crawl	GET	Untuk menjalankan proses crawling	-	duration
Get TF-IDF Document Ranking	/api/v1.0/document_ranking/tf_idf	GET	Untuk mendapatkan ranking halaman menggunakan metode TF IDF	-	keyword, start, length
Get Page-Rank Page Ranking	/api/v1.0/page_ranking/page_rank	GET	Untuk mendapatkan ranking halaman menggunakan metode Page Rank	-	start, length
Get Similarity Overall Ranking	/api/v1.0/overall_ranking/similarity	GET	Untuk mendapatkan ranking halaman secara overall dari pengintegrasian TF IDF dan Page Rank	-	keyword, sort, start, length
Get Crawled Pages	/api/v1.0/crawling/pages	GET	Untuk mendapatkan data halaman yang sudah dicrawling	-	start, length
Get Specific Page Information	/api/v1.0/crawling/page_information	POST	Untuk mendapatkan informasi lengkap halaman dari page id	id_pages	-
Start Insert Crawled Pages	/api/v1.0/crawling/start_insert	POST	Untuk mendapatkan crawl id yang nantinya digunakan untuk memasukkan data halaman ke database crawling	start_urls, keyword, duration_crawl	-
Insert Crawled Page	/api/v1.0/crawling/insert_page	POST	Untuk memasukkan data halaman ke database crawling	page_information, page_forms, page_images, page_linking, page_list, page_scripts, page_styles, page_tables	-

4. REST API

- API untuk menjalankan *crawling*



Search Engine API Production / Run Crawling

GET http://jft.web.id/search-engine/api/v1.0/crawling/start?duration=10

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
duration	10	

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```

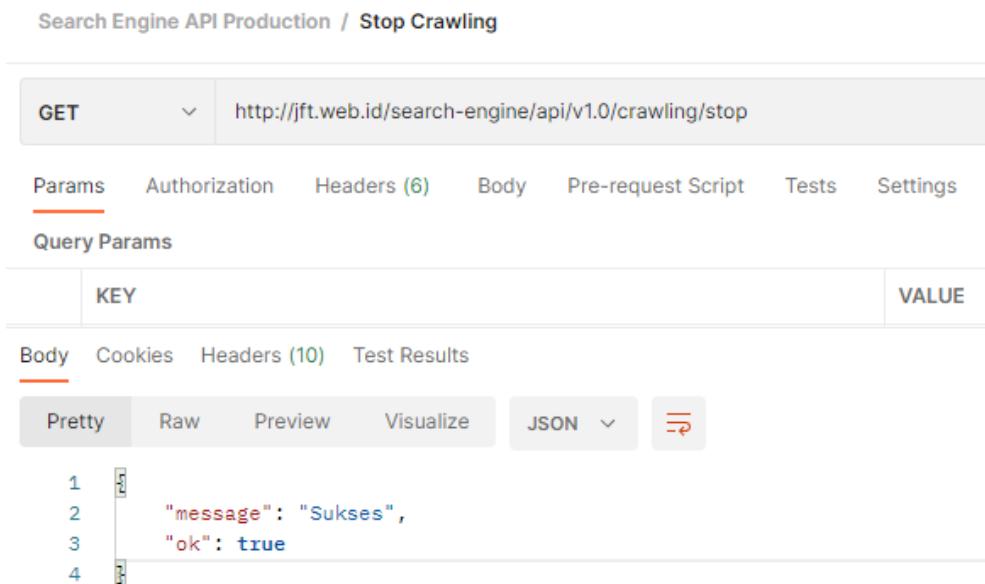
1
2 "message": "Sukses",
3 "ok": true
4

```

Status: 200 OK Time: 104 ms Size: 610 B

Gambar 4.39: API untuk menjalankan *crawling*

- API untuk berhenti *crawling*



Search Engine API Production / Stop Crawling

GET http://jft.web.id/search-engine/api/v1.0/crawling/stop

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
-----	-------

Body Cookies Headers (10) Test Results

Pretty Raw Preview Visualize JSON

```

1
2 "message": "Sukses",
3 "ok": true
4

```

Gambar 4.40: API untuk berhenti *crawling*

- API untuk memulai penambahan halaman dengan data yang sudah ada

Search Engine API Production / Start Insert Crawled Pages

The screenshot shows a Postman interface for a POST request to `http://jft.web.id/search-engine/api/v1.0/crawling/start_insert`. The 'Body' tab is selected, showing a JSON payload:

```
1 {  
2   "start_urls": "https://www.indosport.com",  
3   "keyword": "",  
4   "duration_crawl": 10  
5 }
```

The response body is displayed in the 'Pretty' tab:

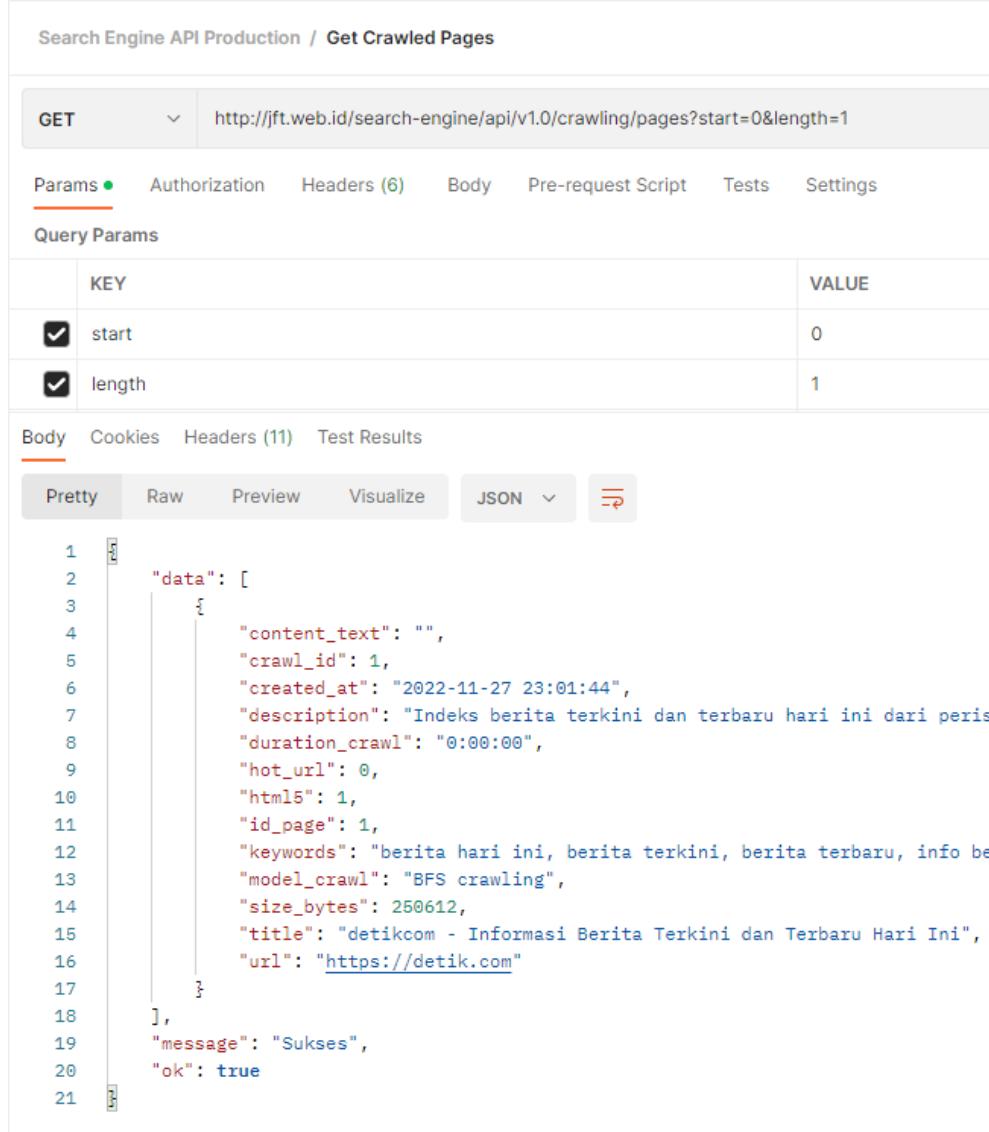
```
1 {  
2   "data": {  
3     "id_crawling": 7  
4   },  
5   "message": "Sukses",  
6   "ok": true  
7 }
```

Gambar 4.41: API untuk memulai penambahan halaman

- *API* untuk menambahkan halaman dengan data yang sudah ada

Gambar 4.42: API untuk penambahan halaman

- API untuk mendapatkan informasi halaman berdasarkan jumlah halaman



The screenshot shows a Postman API request for 'Get Crawled Pages'. The request is a GET method to the URL `http://jft.web.id/search-engine/api/v1.0/crawling/pages?start=0&length=1`. The 'Params' tab is selected, showing two query parameters: 'start' with a value of 0 and 'length' with a value of 1. The 'Body' tab is selected, showing a JSON response with a 'data' array containing one object. The object has fields: content_text (empty string), crawl_id (1), created_at ('2022-11-27 23:01:44'), description ('Indeks berita terkini dan terbaru hari ini dari peris'), duration_crawl ('0:00:00'), hot_url (0), html5 (1), id_page (1), keywords ('berita hari ini, berita terkini, berita terbaru, info be'), model_crawl ('BFS crawling'), size_bytes (250612), title ('detikcom - Informasi Berita Terkini dan Terbaru Hari Ini'), and url ('<https://detik.com>'). The response also includes a message ('Sukses') and an ok field set to true. The JSON response is displayed in a pretty-printed format with line numbers 1 through 21.

```
1
2   "data": [
3     {
4       "content_text": "",
5       "crawl_id": 1,
6       "created_at": "2022-11-27 23:01:44",
7       "description": "Indeks berita terkini dan terbaru hari ini dari peris",
8       "duration_crawl": "0:00:00",
9       "hot_url": 0,
10      "html5": 1,
11      "id_page": 1,
12      "keywords": "berita hari ini, berita terkini, berita terbaru, info be",
13      "model_crawl": "BFS crawling",
14      "size_bytes": 250612,
15      "title": "detikcom - Informasi Berita Terkini dan Terbaru Hari Ini",
16      "url": "https://detik.com"
17    }
18  ],
19  "message": "Sukses",
20  "ok": true
21 ]
```

Gambar 4.43: API untuk mendapatkan informasi halaman berdasarkan jumlah

- *API* untuk mendapatkan informasi halaman berdasarkan *ID* halaman

Search Engine API Production / Get Page Information

POST http://jft.web.id/search-engine/api/v1.0/crawling/page_information

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1
2  .... "id_pages": [5]
3
```

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize **JSON**

```
1  {
2      "data": [
3          {
4              "content_text": "MENU detikcom NEW NEW News Finance Teknologi Entertainment Sport Sepakbola Jatim Jabar Sulsel Sumut Bali Pasang Mata Ads Smart Forum detikEvent Trans Snow World Bunda InsertLive Beautynesia Female Daily CXO Media Home detikFlash e-Flash detik P url telah tercopy Live Chat Minggu 27 Nov 2022 Senin 28 Nov 2022 Selasa 29 Nov 2022 Informasi Jadwal Tidak Ada Informasi Jadwal Tidak Ada Informasi Jadwal Tidak Ada Inf detikcom. All right reserved Kategori Layanan Informasi Jaringan Media",
5              "crawl_id": 1,
6              "created_at": "2022-11-27 23:01:58",
7              "description": "Tonton siaran TV online dari Trans TV, live streaming High Definition (HD) gratis",
8              "duration_crawl": "0:00:06",
9              "hot_url": 0,
10             "html5": 0,
11             "id_page": 5,
12             "keywords": "TransTV, My Trip My Adventure, Insert, Brownnis, Dr OZ Indonesia, Live Stream",
13             "model_crawl": "BFS crawling",
14             "size_bytes": 875801,
15             "title": "Live Streaming Trans TV - Milik Kita Bersama - 20Detik",
16             "url": "https://20.detik.com/live"
17         ],
18         "message": "Sukses",
19         "ok": true
20     }
```

Gambar 4.44: API untuk mendapatkan informasi halaman berdasarkan *ID*

- API untuk mendapatkan *ranking TF IDF*

Search Engine API Production / Get TF-IDF Document Ranking

GET http://jft.web.id/search-engine/api/v1.0/document_ranking/tf_idf?keyword=presiden&start=0&length=2

Params • Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> keyword	presiden
<input checked="" type="checkbox"/> start	0
<input checked="" type="checkbox"/> length	2

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON 

```

1
2   "data": [
3     {
4       "id_tfidf": 3379,
5       "keyword": "presiden",
6       "page_id": 5034,
7       "tfidf_total": 0.3327696799307122,
8       "url": "https://inet.detik.com/cyberlife/d-6426288/kominfo-hapus-situs-presidengoid"
9     },
10    {
11       "id_tfidf": 4721,
12       "keyword": "presiden",
13       "page_id": 5034,
14       "tfidf_total": 0.3327696799307122,
15       "url": "https://inet.detik.com/cyberlife/d-6426288/kominfo-hapus-situs-presidengoid"
16     }
17   ],
18   "message": "Sukses",
19   "ok": true
20

```

Gambar 4.45: API untuk mendapatkan *ranking TF IDF*

- API untuk mendapatkan *ranking PageRank*

Search Engine API Production / Get Page-Rank Page Ranking

GET http://jft.web.id/search-engine/api/v1.0/page_ranking/page_rank?start=0&length=2

Params • Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
<input checked="" type="checkbox"/> start	0
<input checked="" type="checkbox"/> length	2

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON

```
1
2   "data": [
3     {
4       "id_pagerank": 30,
5       "page_id": 30,
6       "pagerank_score": 0.0013682586525913308,
7       "url": "https://adsmart.detik.com"
8     },
9     {
10       "id_pagerank": 40,
11       "page_id": 40,
12       "pagerank_score": 0.001364460785806049,
13       "url": "https://www.detik.com"
14     }
15   ],
16   "message": "Sukses",
17   "ok": true
18 ]
```

Gambar 4.46: API untuk mendapatkan *ranking PageRank*

- API untuk mendapatkan *ranking* berdasarkan *keyword* secara keseluruhan

Search Engine API Production / Get Similarity Overall Ranking

GET http://jft.web.id/search-engine/api/v1.0/overall_ranking/similarity?keyword=presiden&sort=similarity&start=0&length=3

Params (6) Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE
<input checked="" type="checkbox"/>	keyword	presiden
<input checked="" type="checkbox"/>	sort	similarity
<input checked="" type="checkbox"/>	start	0
<input checked="" type="checkbox"/>	length	3

Body Cookies Headers (11) Test Results

Pretty Raw Preview Visualize JSON [-P](#)

```

1
2   "data": [
3     {
4       "id_page": 5034,
5       "pagerank_score": 1.5033663234404962e-05,
6       "similarity_score": 0.1996678214237211,
7       "tfidf_total": 0.3327696799307122,
8       "url": "https://inet.detik.com/cyberlife/d-6425288/kominfo-hapus-situs-presidengoid"
9     },
10    {
11      "id_page": 5034,
12      "pagerank_score": 1.5033663234404962e-05,
13      "similarity_score": 0.1996678214237211,
14      "tfidf_total": 0.3327696799307122,
15      "url": "https://inet.detik.com/cyberlife/d-6425288/kominfo-hapus-situs-presidengoid"
16    },
17    {
18      "id_page": 5034,
19      "pagerank_score": 1.5033663234404962e-05,
20      "similarity_score": 0.1996678214237211,
21      "tfidf_total": 0.3327696799307122,
22      "url": "https://inet.detik.com/cyberlife/d-6425288/kominfo-hapus-situs-presidengoid"
23    }
24  ],
25  "message": "Sukses",
26  "ok": true
27

```

Gambar 4.47: API untuk mendapatkan *ranking* secara keseluruhan

5. Program *search engine* berbasis *console*



```
[root@jft search-engine]# /opt/rh/rh-python38/root/usr/bin/python run_search_engine_console.py
Input keyword pencarian: resesi 2023
Hasil pencarian:
1. Apa Itu Resesi? Begini Dampak dan Kiat Menghadapinya
https://www.detik.com/edu/detikpedia/d-6417781/apa-itu-resesi-begini-dampak-dan-kiat-menghadapinya
Trisna Wulandari - detikEdu Senin,21 Nov 2022 15:30 WIB 0 komentar URL telah disalin Dampak resesi

2. Podcast: Anti Panic Panic Club Ramalan Ekonomi Gelap 2023
https://finance.detik.com/berita-ekonomi-bisnis/d-6361031/podcast-anti-panic-panic-club-ramalan-ekonomi-gelap-2023
Tolak Miskin Podcast: Anti Panic Panic Club Ramalan Ekonomi Gelap 2023 Eduardo Simorangkir - detikFi

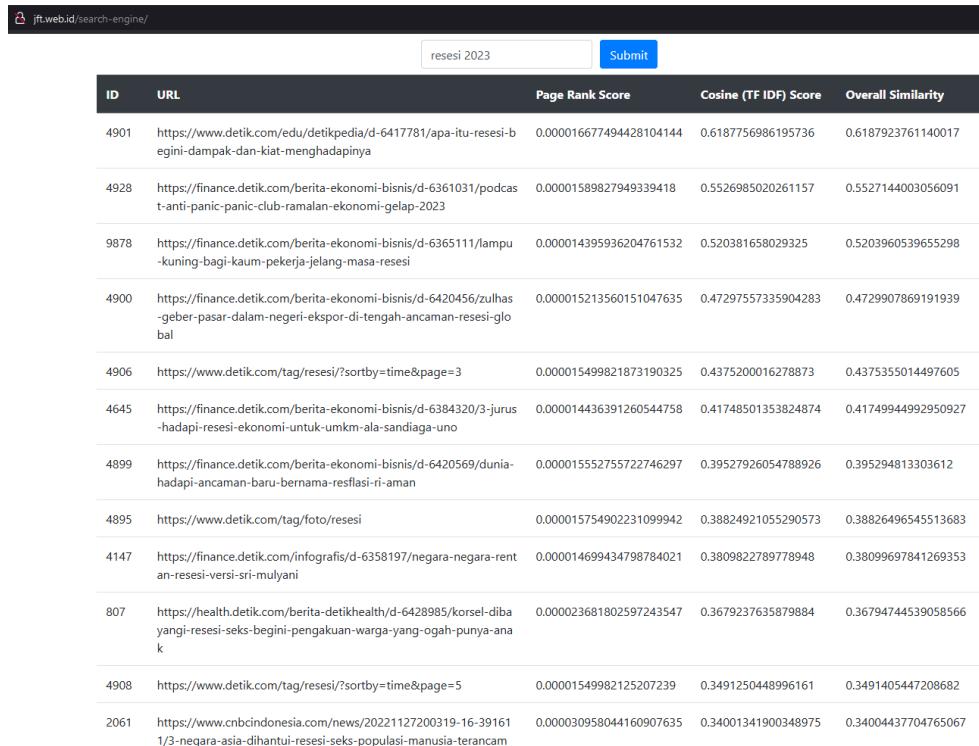
3. Lampu Kuning Bagi Kaum Pekerja Jelang Masa Resesi
https://finance.detik.com/berita-ekonomi-bisnis/d-6365111/lampu-kuning-bagi-kaum-pekerja-jelang-masa-resesi
Lampu Kuning Bagi Kaum Pekerja Jelang Masa Resesi Edward. F. Kusuma - detikFinance Senin,24 Okt 2022

4. Zulhas Geber Pasar Dalam Negeri-Eksport di Tengah Ancaman Resesi Global
https://finance.detik.com/berita-ekonomi-bisnis/d-6420456/zulhas-geber-pasar-dalam-negeri-ekspor-di-tengah-ancaman-resesi-global
Zulhas Geber Pasar Dalam Negeri-Eksport di Tengah Ancaman Resesi Global Mega Putra Ratya - detikFinan

5. Berita dan Informasi Resesi Terkini dan Terbaru Hari ini - detikcom
https://www.detik.com/tag/resesi/?sortby=time&page=3
detikFinance Selasa,15 Nov 2022 08:30 WIB Ekonomi Global Dihantui Resesi,Investasi Apa yang Masih Bi
```

Gambar 4.48: Program *search engine* berbasis *console*

6. Tampilan web sederhana (*front-end*) yang menampilkan hasil relevan beserta dengan skornya



ID	URL	Page Rank Score	Cosine (TF IDF) Score	Overall Similarity
4901	https://www.detik.com/edu/detikpedia/d-6417781/apa-itu-resesi-begini-dampak-dan-kiat-menghadapinya	0.000016677494428104144	0.6187756986195736	0.6187923761140017
4928	https://finance.detik.com/berita-ekonomi-bisnis/d-6361031/podcast-anti-panic-panic-club-ramalan-ekonomi-gelap-2023	0.00001589827949339418	0.5526985020261157	0.5527144003056091
9878	https://finance.detik.com/berita-ekonomi-bisnis/d-6365111/lampu-kuning-bagi-kaum-pekerja-jelang-masa-resesi	0.000014395936204761532	0.520381658029325	0.5203960539655298
4900	https://finance.detik.com/berita-ekonomi-bisnis/d-6420456/zulhas-geber-pasar-dalam-negeri-ekspor-di-tengah-ancaman-resesi-global	0.000015213560151047635	0.47297557335904283	0.4729907869191939
4906	https://www.detik.com/tag/resesi/?sortby=time&page=3	0.000015499821873190325	0.4375200016278873	0.4375355014497605
4645	https://finance.detik.com/berita-ekonomi-bisnis/d-6384320/3-jurus-hadapi-resesi-ekonomi-untuk-umkm-ala-sandiaga-uno	0.000014436391260544758	0.41748501353824874	0.41749944992950927
4899	https://finance.detik.com/berita-ekonomi-bisnis/d-6420569/dunia-hadapi-ancaman-baru-bernama-resfiasi-ri-aman	0.000015552755722746297	0.39527926054788926	0.395294813303612
4895	https://www.detik.com/tag/foto/resesi	0.000015754902231099942	0.38824921055290573	0.38826496545513683
4147	https://finance.detik.com/infografis/d-6358197/negara-negara-rentan-resesi-versi-sri-mulyani	0.000014699434798784021	0.3809822789778948	0.3809697841269353
807	https://health.detik.com/berita-detikhealth/d-6428985/korsel-dibayangi-resesi-seks-begini-pengakuan-warga-yang-ogah-punya-ank	0.000023681802597243547	0.3679237635879884	0.36794744539058566
4908	https://www.detik.com/tag/resesi/?sortby=time&page=5	0.00001549982125207239	0.3491250448996161	0.3491405447208682
2061	https://www.cnbcindonesia.com/news/20221127200319-16-391611/3-negara-asia-dihantui-resesi-seks-populasi-manusia-terancam	0.000030958044160907635	0.34001341900348975	0.34004437704765067

Gambar 4.49: Tampilan web sederhana *search engine*

7. *Source code* dan dokumentasi lengkap

The screenshot shows a GitHub repository page for 'lazuardyk/search-engine'. At the top, there is a list of commits:

Commit	Description	Time Ago
run_search_engine_console.py	add main frontend, and fix se console	12 days ago
run_similarity.py	create search engine console and improve others	21 days ago
run_tf_idf.py	create search engine console and improve others	21 days ago
search-engine.wsgi	update deployment	10 days ago

Below the commits is the README.md file content:

Search Engine

Aplikasi search engine yang dibuat dengan menggunakan crawler, document ranking, dan page ranking

Cara Penggunaan

1. Pastikan komputer/server sudah terinstall Python 3.6+ dan MySQL
2. Buka file `.env` dan ubah konfigurasinya dengan benar (akses database, konfigurasi crawler, dll)
3. Install library python yang diperlukan dengan menjalankan `pip install -r requirements.txt`
4. Jalankan program sesuai dengan perintah di bawah

Perintah

General

- Python `run_crawl.py` untuk menjalankan crawler
- Python `run_page_rank.py` untuk menjalankan page rank
- Python `run_tf_idf.py` untuk menjalankan tf idf
- Python `run_api.py` untuk menjalankan REST API

Background Services

- Gunakan `crawl.service` di folder services untuk menjalankan crawler, page rank, dan tf idf di background menggunakan `systemd` pada server

File Dokumentasi

- Entity Relationship Diagram (ERD)
- Class Diagram

Gambar 4.50: *Source code* dan dokumentasi di Github

Source code keseluruhan arsitektur *search engine* dan dokumentasi lengkap beserta cara penggunaannya terdapat di Github, yaitu pada *link* <https://github.com/lazuardyk/search-engine>

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian arsitektur *search engine* yang telah dirancang, maka diperoleh kesimpulan sebagai berikut:

1. Perancangan arsitektur *search engine* yang mengintegrasikan *web crawler*, algoritma *page ranking*, dan *document ranking* dirancang menggunakan metode pengembangan Scrum.
2. Arsitektur *search engine* yang dibuat mencakup struktur *project*, rancangan diagram, konfigurasi *server*, *web service* berupa *REST API*, program *console* dan tampilan web sederhana, serta dokumentasi lengkap *project*.
3. Berdasarkan hasil pengujian, didapatkan bahwa fungsi-fungsi yang terdapat pada sistem berjalan seluruhnya dengan baik.

B. Saran

Adapun saran untuk penelitian selanjutnya antara lain:

1. Untuk penelitian selanjutnya agar membuat tampilan atau *frontend website* yang menarik dan mudah untuk digunakan.
2. Meningkatkan kinerja *web crawler* agar memiliki penggunaan *RAM* yang lebih efisien dan dapat menyimpan dokumen yang berbentuk *file*.
3. Meningkatkan kinerja *PageRank* atau *TF IDF* agar proses perankingan menjadi lebih cepat.

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.
- Firdaus, Pasnur, dan Wabdillah (2019). Implementasi cosine similarity untuk peningkatan akurasi pengukuran kesamaan dokumen pada klasifikasi dokumen berita dengan k nearest neighbour. *Inspiration: Jurnal Teknologi Informasi dan Komunikasi*, 9.
- 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.
- Roul, R., Sahoo, J., dan Arora, K. (2019). *Query-Optimized PageRank: A Novel Approach*, pages 673–683.
- 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.

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 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.

LAMPIRAN B

Dokumentasi *Testing*



LAMPIRAN C

Kode untuk Uji Keandalan Server

```
import time
import requests

total_time = 0
n = 1000 # jumlah iterasi
for i in range(n):
    start = time.perf_counter()
    url = "http://jft.web.id/search-engine/api/v1.0/
overall_ranking/similarity?keyword=presiden
&sort=similarity&start=0&length=100"
    response = requests.get(url)
    request_time = time.perf_counter() - start
    total_time += request_time
    print(total_time, request_time, i)
print("~")
print(total_time / n)
```

DAFTAR RIWAYAT HIDUP



Lazuardy Khatulistiwa, lahir di Jakarta pada tanggal 18 Oktober 2000. Penulis merupakan anak ketiga dari tiga bersaudara yaitu dari pasangan Efdy Tawara dan Lely Fawziah. Penulis saat ini tinggal di Jl. Waru Doyong RT. 007. RW.008, Jakarta Timur, DKI Jakarta, Indonesia 13930.

No. Ponsel: 081395863983

E-mail: lazdevs@gmail.com

Riwayat Pendidikan: Penulis mengenyam pendidikan dan lulus dari sekolah dasar di SDN 08 Jakarta pada tahun 2006 – 2012, kemudian melanjutkan pendidikan ke SMPN 27 Jakarta pada tahun 2012 – 2015. Setelah itu, penulis melanjutkan pendidikan di SMAN 59 Jakarta pada tahun 2015 – 2018. Di tahun yang sama (2018), penulis diterima kuliah di Universitas Negeri Jakarta (UNJ) Fakultas Matematika dan Ilmu Pengetahuan Alam (FMIPA) dengan program studi Ilmu Komputer dengan jalur masuk melalui SNMPTN.