

**IMPLEMENTASI DETEKSI SIMILARITAS KODE PADA
SISTEM PRAKTIKUM PEMROGRAMAN WEB BERBASIS
UNIT TESTING JAVA SCRIPT**



TUGAS AKHIR

Disusun dalam rangka memenuhi salah satu persyaratan

Untuk menyelesaikan program Strata-1 Departemen Teknik Informatika

Fakultas Teknik Universitas Hasanuddin

Makassar

Disusun Oleh:

UMNIYAH NUR APRILYAH

D421 15 016

DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

MAKASSAR

2020

LEMBAR PENGESAHAN

“IMPLEMENTASI DETEKSI SIMILARITAS KODE PADA SISTEM PRAKTIKUM PEMROGRAMAN WEB BERBASIS UNIT TESTING JAVA SCRIPT”

Disusun Oleh:


UMNIYAH NUR APRILYAH
D421 15 016

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 11 November 2020. Diterima dan disahkan sebagai salah satu syarat memperoleh gelar sarjana Teknik (S.T) pada Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.


Makassar, 11 November 2020

Disetujui Oleh:

Pembimbing I,


Dr. Eng. Muhammad Niswar, ST., M.IT
NIP. 19750716 200212 1 004

Pembimbing II,


Iqra Aswad, ST., M.T
NIP. 19901128 201904 3 001

Diterima dan disahkan oleh:
Ketua Departemen S1 Teknik Informatika



Dr. Amir Ahmad Ilham, S.T., M.IT
NIP. 19731010 199802 1 001

SURAT PERNYATAAN

Saya yang bertanda tangan dibawah ini :

N a m : Umniyah Nur Aprilyah

N I M : D42115016

Judul Skripsi : Implementasi Deteksi Similaritas Kode pada Sistem
Praktikum Pemrograman Web Berbasis Unit Testing Java
Script

Bahwa benar ada Karya Ilmiah Saya dan bebas dari Plagianisme (duplikasi).
Demikianlah Surat Pernyataan ini dibuat, jika dikemudian hari ditemukan bukti
ketidakaslilan atas Karya Ilmiah ini maka Saya bersedia mempertanggungjawabkan
sesuai Peraturan Perundang-Undangan yang berlaku.

Makassar, 11 November 2020

Yang Bersangkutan



Umnayah Nur Aprilyah

ABSTRACT

Practicum assignment is a standardization which is made to measure the success and the level of students' understanding of a competency in the learning process. Currently, the teaching and learning process of students who taking web programming courses in Informatics Engineering Hasanuddin University can be done by online on the practicum module web. Unfortunately, sending assignment online opens the opportunities for collaboration or copying practicum assignments between students. Currently, there is no checking code similarity in practicum assignments, so there are many student practicum assignments that have similarities. To prevent these problems, additional facilities will be applied to detect the existence of similarity of assignment or code similarities to student programming assignment on the practicum web module. By using Jaro Winkler algorithm, a system that can detect the similarity of tasks or code similarity in student assignments on the web model of the practicum can be created. The data taken from student assignments that have previously been stored in the practicum module web database, the process of checking student assignments in this system used Esprima. The method used Lexical Analysis or Tokenizing in Esprima. After the Tokenizing results came out, the similarity level will be checked using the Jaro Winkler algorithm. The results of calculations performed by the system and calculations done manually, obtain the percentage as much as 100%.

Keywords: practicum assignment, code similarity, practicum module web, esprima, jaro winkler

ABSTRAK

Tugas Praktikum merupakan standarisasi yang dibuat untuk mengukur keberhasilan dan tingkat pemahaman mahasiswa dari sebuah kompetensi dalam proses belajar. Saat ini proses belajar mengajar mahasiswa yang mengambil mata kuliah pemrograman *web* Teknik Informatika Universitas Hasanuddin bisa dilakukan secara *online* pada *web* modul praktikum. Hanya saja, pengiriman tugas secara *online* membuka peluang munculnya kerjasama atau menyalin tugas praktikum antar mahasiswa. Saat ini belum ada pengecekan similaritas kode pada tugas praktikum, sehingga banyak tugas praktikum mahasiswa yang memiliki kemiripan. Untuk mencegah permasalahan tersebut, maka akan diterapkan fasilitas tambahan untuk mendeteksi adanya kesamaan tugas atau similaritas kode terhadap tugas pemrograman mahasiswa pada *web* modul praktikum tersebut. Dengan menggunakan Algoritma *Jaro Winkler*, maka dapat dibuat sistem yang dapat mendeteksi kesamaan tugas atau similaritas kode pada tugas-tugas mahasiswa yang berada pada *web* modul praktikum tersebut. Adapun data yang digunakan adalah tugas mahasiswa yang sebelumnya telah tersimpan ke dalam *database web* modul praktikum, proses pengecekan tugas mahasiswa di dalam sistem ini menggunakan *Esprima*. Metode yang digunakan adalah *lexical analysis* atau *tokenizing* yang terdapat pada *Esprima*, setelah hasil *tokenizing* keluar maka akan di cek tingkat kesamaannya menggunakan Algoritma *Jaro Winkler*. Untuk hasil perhitungan yang dilakukan oleh sistem dan perhitungan yang dilakukan secara manual didapatkan persentase sebesar 100%.

Kata kunci: tugas praktikum, similaritas kode, web modul praktikum, *esprima*, *Jaro Winkler*

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Allah SWT karena atas berkat Rahmat dan Karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **“IMPLEMENTASI DETEKSI SIMILARITAS KODE PADA SISTEM PRAKTIKUM PEMROGRAMAN WEB BERBASIS UNIT TESTING JAVA SCRIPT”** ini dapat diselesaikan sebagai salah satu syarat dalam menyelesaikan jenjang Strata-1 Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin.

Penulis menyadari bahwa dalam penyusunan dan penulisan Tugas Akhir ini tidak lepas dari bantuan, bimbingan serta dukungan dari berbagai pihak, dari masa perkuliahan sampai dengan penyusunan tugas akhir, sehingga pada kesempatan ini penulis menyampaikan ucapan terima kasih sedalam-dalamnya kepada:

1. Allah SWT atas semua berkat, karunia serta pertolongan-Nya yang telah diberikan kepada kami disetiap langkah dalam pembuatan program hingga penulisan laporan skripsi ini;
2. Orang tua penulis, Bapak Drs. Muhaemi Pance M.Si dan Ibu Amaliah Abidin SS dan Adek Shafwan Aprilyan, yang selalu memberikan dukungan, doa, dan semangat kepada penulis;
3. Bapak Dr. Eng. Muhammad Niswar, ST., M.I.T selaku pembimbing 1 dan Bapak Iqra Aswad, ST.,M.T selaku pembimbing II yang selalu menyediakan waktu, tenaga, pikiran dan perhatian yang luar biasa untuk mengarahkan penulis dalam penyusunan tugas akhir;

4. Bapak Amil Ahmad Ilham, ST., M.I.T dan Bapak A. Ais Prayogi Alimuddin, S.T., M.Eng yang telah memberikan bimbingan dan masukan sehingga laporan skripsi ini menjadi lebih baik;
5. Bapak Amil Ahmad Ilham, ST., M.I.T selaku Ketua Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin atas bimbingannya selama masa perkuliahan penulis;
6. Dosen dan Staf Departemen Teknik Informatika Fakultas Teknik Universitas Hasanuddin yang telah membantu penulis;
7. Teman-teman penulis (Fadel Rezky Ramadhan, dan Laura Natalia Nainggolan, ST) yang telah memberikan banyak bantuan dalam proses penyelesaian Tugas Akhir ini;
8. Teman-teman Khayangan (Ammy, Ardy, Dian, Fira, Kiky), yang telah memberikan semangat dan menjadi teman penulis selama menjalani perkuliahan sampai dengan penyelesaian Tugas Akhir.
9. Teman-teman Sapu Lidi (Ai, Ammy, Arief, Charina, Fadel, Fahril, Khusnul, Laura, Reka, Ryan, Sabtian, dan Said), yang telah memberikan semangat dan menjadi teman penulis selama menjalani perkuliahan sampai dengan penyelesaian Tugas Akhir.
10. Teman-teman dan kakak-kakak Laboratorium UBICON Unhas yang telah memberikan begitu banyak bantuan selama penelitian dan diskusi progress penyusunan Tugas Akhir;
11. Teman-teman Hypervisor FT UH atas dukungan dan semangat yang diberikan selama ini;

12. Keluarga besar GKM Al-Muhandis FT-UH terkhusus Departemen Kemuslimahan yang telah banyak memberikan nasehat dan dukungan selama penulisan tugas akhir ini.
13. Keluarga besar Genetis Smadab atas dukungan dan semangat yang diberikan selama ini;
14. Fuad Khairi Hamid, ST yang telah memberikan dukungan dan bantuan dalam proses penyelesaian Tugas Akhir ini;
15. Seluruh pihak yang tidak sempat kami sebutkan satu persatu yang telah banyak meluangkan tenaga, waktu, dan pikiran yang tanpa sadar telah menjadi inspirasi penulis.

Akhir kata, penulis berharap semoga Allah SWT. berkenan membalas segala kebaikan dari semua pihak yang telah banyak membantu. Semoga Tugas Akhir ini dapat memberikan manfaat bagi para pembaca.

Makassar, 11 November 2020

Penulis

DAFTAR ISI

IMPLEMENTASI DETEKSI SIMILARITAS KODE PADA SISTEM PRAKTIKUM PEMROGRAMAN WEB BERBASIS UNIT TESTING JAVA SCRIPT	i
LEMBAR PENGESAHAN	ii
SURAT PERNYATAAN	iii
ABSTRACT	iv
ABSTRAK	v
KATA PENGANTAR	vi
DAFTAR ISI	ix
DAFTAR GAMBAR	xii
DAFTAR TABEL	xiv
BAB I PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	2
1.3. Tujuan Penelitian	3
1.4. Manfaat Penelitian	3
1.5. Batasan Masalah	3
1.6. Sistematika Penulisan	4
BAB II TINJAUAN PUSTAKA	6
2.1. Sistem Praktikum	6
2.2. Pengertian Similarity	7
2.2.1. Algoritma	8
2.2.2. Jaro-Winkler	12
2.3. Bahasa Pemrograman	19
2.3.1. HTML	19

2.3.2.	PUG	20
2.3.3.	CSS	22
2.3.4.	JavaScript	23
2.4.	Database	31
2.4.1.	Struktur Data	32
2.4.2.	MongoDB	32
2.4.3.	NoSQL	33
2.4.4.	Redis	35
2.5.	Hackathon Starter Pack.....	35
2.6.	Sublime Text	36
BAB III METODOLOGI PENELITIAN		37
3.1.	Tahapan Penelitian.....	37
3.2.	Waktu dan Lokasi Penelitian	38
3.3.	Instrumen Penelitian	39
3.4.	Data yang digunakan	39
3.5.	Gambaran Umum Sistem	40
3.5.1.	Rancangan Umum Sistem	40
3.5.2.	Proses Kerja Algoritma Jaro-Winkler.....	43
3.5.3.	Database Diagram Sistem.....	46
BAB IV HASIL DAN PEMBAHASAN.....		49
4.1.	Penerapan Sistem Similaritas Kode.....	49
4.1.1.	Pengambilan Data dari Database	49
4.1.2.	Analisis Leksikal Program (Tokenizing) menggunakan Esprima	50
4.1.3.	Pengecekan Similarity menggunakan Algoritma Jaro-Winkler	53
4.1.4.	Pengelompokan Data oleh Sistem	71

BAB V PENUTUP	73
5.1. Kesimpulan	73
5.2. Saran	73
DAFTAR PUSTAKA	74
LAMPIRAN	77

DAFTAR GAMBAR

Gambar 2.1	Grafik Rata-rata Kecepatan Proses Algoritma	18
Gambar 2.2	Grafik Rata-rata Hasil Perbandingan Algoritma	19
Gambar 2.3	Syntax template PUG (disimpan dengan ekstensi .pug).....	21
Gambar 2.4	Kode menghasilkan halaman HTML.....	21
Gambar 2.5	Contoh Penggunaan Tokenizer Esprima	30
Gambar 3.1	Diagram Alur	37
Gambar 3.2	Rancangan Umum Sistem.....	41
Gambar 3.3	Proses Kerja Alogrima Jaro-Winkler.....	44
Gambar 3.4	Database Diagram Sistem.....	47
Gambar 4.1	Potongan Kode Esprima	51
Gambar 4.2	Proses konversi hasil tokenisasi kategori ke dalam bentuk karakter	52
Gambar 4.3	Keterangan kategori proses konversi	52
Gambar 4.4	Tampilan Hasil Esprima pada Sistem.....	53
Gambar 4.5	Tampilan Hasil Similarity untuk dua data mahasiswa	55
Gambar 4.6	Tampilan Hasil Similarity untuk dua data mahasiswa	56
Gambar 4.7	Tampilan Hasil Similarity untuk dua data mahasiswa	57
Gambar 4.8	Tampilan Hasil Similarity untuk dua data mahasiswa	59
Gambar 4.9	Tampilan Hasil Similarity untuk dua data mahasiswa	61
Gambar 4.10	Tampilan Hasil Similarity untuk dua data mahasiswa	62
Gambar 4.11	Tampilan Hasil Similarity untuk dua data mahasiswa	64
Gambar 4.12	Tampilan Hasil Similarity untuk dua data mahasiswa	65
Gambar 4.13	Tampilan Hasil Similarity untuk dua data mahasiswa	67

Gambar 4.14 Tampilan Hasil Similarity untuk dua data mahasiswa 69

Gambar 4.15 Tampilan Hasil Data Similarity yang telah dikelompokkan..... 71

DAFTAR TABEL

Tabel 2.1 Daftar Konfigurasi fungsi parseScript/parseModel.....	28
Tabel 2.2 Daftar Konfigurasi pada proses tokenis	29
Tabel 4.1 Data Tugas Mahasiswa.....	49
Tabel 4.2 Perbandingan tugas mahasiswa yang telah diesprimakan.....	53

BAB I

PENDAHULUAN

1.1. Latar Belakang

Tugas Praktikum merupakan standarisasi yang dibuat untuk mengukur keberhasilan dan tingkat pemahaman mahasiswa dari sebuah kompetensi dalam proses belajar. Sistem Pengumpulan Tugas Praktikum merupakan tempat untuk mengirim dokumen tugas mahasiswa. Hanya saja, pengiriman tugas secara *online* membuka peluang munculnya kerjasama atau menyalin tugas praktikum antar mahasiswa.

Saat ini belum ada pengecekan kesamaan kode pada tugas praktikum, sehingga banyak tugas praktikum mahasiswa yang memiliki kemiripan. Berdasarkan permasalahan tersebut, maka akan diterapkan fasilitas tambahan untuk mendeteksi kesamaan tugas mahasiswa berbasis *unit testing* pada sistem tugas praktikum yang dikirim mahasiswa.

Sebuah salinan elektronik berdasarkan Undang-Undang Perguruan Tinggi nomer 12 tahun 2012, pasal 31 tentang Pendidikan Jarak Jauh (PJJ) menjelaskan bahwa PJJ merupakan proses belajar mengajar yang dilakukan secara jarak jauh melalui penggunaan berbagai media komunikasi. PJJ adalah suatu sistem pendidikan yang memiliki karakteristik terbuka, belajar mandiri, dan belajar tuntas dengan memanfaatkan Teknologi Informasi & Komunikasi (TIK) dan/atau menggunakan teknologi lainnya. Untuk mendukung terlaksananya program PJJ, maka *Learning Management System* (LMS) dengan nama SPELL IDE dibuat. LMS

ini digunakan untuk mengelola pembelajaran jarak jauh secara *online*. Salah satu fitur yang ada pada *website* SPELL IDE adalah *course* dimana pada fitur ini disetiap pembelajarannya terdapat tugas yang akan di *upload* oleh mahasiswa. Selanjutnya dosen akan memeriksa hasil tugas mahasiswa secara manual. *Upload* tugas berupa dokumen ini membuka peluang bagi para mahasiswa untuk melakukan tindakan kecurangan seperti menyalin tugas temannya. Saat ini pengecekan terkait kemiripan tugas antara mahasiswa pada *website* SPELL IDE masih dilakukan secara manual.

Pendeteksi ini akan diterapkan pada tugas praktikum pemrograman *web* untuk mahasiswa Teknik Informatika, Universitas Hasanuddin. Oleh karena itu, judul penelitian yang diajukan "Implementasi Deteksi Similaritas Kode pada Sistem Praktikum Pemrograman Web berbasis Unit Testing Javascript".

1.2. Rumusan Masalah

Rumusan masalah pada tugas akhir ini adalah:

1. Bagaimana merancang sistem praktikum pemrograman *web* yang dapat mengetahui mahasiswa melakukan tindakan kerjasama atau menyalin pada tugas yang diberikan?
2. Bagaimana mendeteksi kesamaan kode pada sistem praktikum pemrograman *web* berbasis *unit testing javascript*?
3. Bagaimana hasil pengujian deteksi kesamaan kode pada sistem praktikum pemrograman *web* berbasis *unit testing javascript*?

1.3. Tujuan Penelitian

Tujuan akhir dari penelitian ini adalah:

1. Merancang sistem praktikum pemrograman *web* yang dapat mengetahui mahasiswa melakukan tindakan kerjasama atau menyalin pada tugas yang diberikan.
2. Mendeteksi kesamaan kode pada sistem praktikum pemrograman *web* berbasis *unit testing javascript*.
3. Mengetahui hasil pengujian deteksi kesamaan kode pada sistem praktikum pemrograman *web* berbasis *unit testing javascript*.

1.4. Manfaat Penelitian

Dengan adanya penelitian ini, diharapkan manfaat yang didapatkan adalah:

1. Sistem ini dapat mempermudah dosen dalam memonitoring dan mengevaluasi mahasiswa dalam mengerjakan tugas yang diberikan.
2. Mahasiswa dapat mengimplementasikan *unit testing* pada program.

1.5. Batasan Masalah

Yang menjadi batasan masalah dalam tugas akhir ini adalah:

1. Sistem yang dibuat berbasis *website* menggunakan *javascript*.
2. Menggunakan *mocha test*.
3. Digunakan untuk mahasiswa yang mengambil mata kuliah pemrograman *web* Universitas Hasanuddin.

4. Dalam penilaian *similarity* kode tugas praktikum dibatasi oleh beberapa kategori yang diberi simbol dengan warna. (Hijau : 50% -89%, Merah : 90% - 100%).
5. Menggunakan algoritma *similarity text* (*Algoritma Jaro Winkler*).

1.6. Sistematika Penulisan

Untuk memberikan gambaran singkat mengenai isi tulisan secara keseluruhan, maka akan diuraikan beberapa tahapan dari penulisan secara sistematis, yaitu :

BAB I PENDAHULUAN

Bab ini menguraikan secara umum mengenai hal yang menyangkut latar belakang, perumusan masalah dan batasan masalah, tujuan, manfaat, dan sistematika penulisan.

BAB II TINJAUAN PUSTAKA

Bab ini berisi teori-teori tentang hal-hal yang berhubungan dengan sistem yang dibuat, antara lain mengenai modul praktikum, *similarity*, metode yang digunakan, hingga bahasa dan *framework* yang digunakan dalam pengembangan sistem.

BAB III METODOLOGI PENELITIAN

Bab ini berisi analisis kebutuhan sistem, tahapan penelitian, instrument penelitian, perancangan system, penerapan algoritma serta proses kerja sistem secara keseluruhan.

BAB IV HASIL DAN PEMBAHASAN

Bab ini berisi tentang hasil unjuk kerja sistem serta analisa dan pembahasan mengenai hasil kerja sistem yang disertai tabel hasil penelitian.

BAB V PENUTUP

Bab ini berisi kesimpulan yang didapatkan berdasarkan hasil penelitian yang telah dilakukan serta saran-saran untuk pengembangan lebih lanjut.

BAB II

TINJAUAN PUSTAKA

2.1. Sistem Praktikum

Pratikum berasal dari kata praktik yang artinya pelaksanaan secara nyata apa yang disebut dalam teori. Sedangkan praktikum adalah bagian dari pengajaran yang bertujuan agar siswa mendapat kesempatan untuk menguji dan melaksanakan di keadaan nyata, apa yang diperoleh dari teori dan pelajaran praktik (KBBI, 2001:785). Menurut Sudirman (1992:163) metode praktikum adalah cara penyajian pelajaran kepada siswa untuk melakukan percobaan dengan mengalami dan membuktikan sesuatu yang dipelajari.

Melalui praktikum, peserta didik dapat memiliki banyak pengalaman, baik berupa pengamatan langsung atau bahkan melakukan percobaan sendiri dengan objek tertentu. Tidak diragukan lagi bahwa melalui pengalaman langsung (*first-hand experiences*), peserta didik dapat belajar lebih mudah dibandingkan dengan belajar melalui sumber sekunder, misalnya buku. Hal tersebut sangat sesuai dengan pendapat Bruner yang menyatakan bahwa anak belajar dengan pola *inactive* melalui perbuatan (*learning by doing*) akan dapat mentransfer ilmu pengetahuan yang dimilikinya pada berbagai situasi (Tresna Sastrawijaya, 1998 : 17).

Sistem Praktikum merupakan standarisasi yang dibuat untuk mengukur keberhasilan dan tingkat pemahaman mahasiswa dari sebuah kompetensi dalam proses belajar. Sistem Pengumpulan Tugas Praktikum merupakan suatu sistem pendidikan untuk mengirimkan tugas mahasiswa, memiliki karakteristik terbuka, belajar mandiri, dan belajar tuntas dengan memanfaatkan teknologi. Sistem ini

digunakan untuk mengelola pembelajaran secara *online*. Salah satu fitur yang ada pada sistem praktikum ini adalah tugas di *upload* oleh mahasiswa kemudian selanjutnya dosen akan memeriksa hasil tugas mahasiswa yang telah dikirim ke sistem praktikum tersebut.

2.2. Pengertian Similarity

Konsep *similarity* sudah menjadi isu yang sangat penting di hampir setiap bidang ilmu pengetahuan. Menurut Zaka, (2009) dalam disertasinya menjelaskan tiga macam teknik yang dibangun untuk menentukan nilai *similarity*.

- a. *Distance-based similarity measure* mengukur tingkat kesamaan dua buah objek dari segi jarak geometris dari variabel-variabel yang tercakup di dalam kedua objek tersebut. Metode *Distance-based similarity* ini meliputi *Minkowski Distance*, *Manhattan/City block distance*, *Euclidean distance*, *Jaccard distance*, *Dice's Coefficient*, *Cosine similarity*, *Levenshtein Distance*, *Hamming distance*, dan *Soundex distance*.
- b. *Feature-based similarity measure* melakukan penghitungan tingkat kemiripan dengan merepresentasikan objek ke dalam bentuk *feature-feature* yang ingin di perbandingkan. *Feature-based similarity measure* banyak digunakan dalam melakukan pengklasifikasian atau *pattern matching* untuk gambar dan teks.
- c. *Probabilistic-based similarity measure* menghitung tingkat kemiripan dua objek dengan merepresentasikan dua set objek yang dibandingkan dalam bentuk *probability*. *Kullback Leibler Distance* dan *Posterior Probability* termasuk dalam metode ini.

Menurut Mutiara dalam Surahman, (2013) untuk menentukan jenis kemiripan antara dokumen yang diuji ada 5 jenis penilaian persentase :

- a) 0% : Hasil uji 0% berarti kedua dokumen tersebut benar-benar berbeda baik dari segi isi dan kalimat secara keseluruhan atau tidak ada kemiripan.
- b) < 15%: Hasil uji 15% berarti kedua dokumen tersebut hanya mempunyai sedikit kemiripan.
- c) 15% - 50%: Hasil uji 15% - 50% berarti menandakan dokumen tersebut termasuk mendekati kemiripan.
- d) >50% : Hasil uji lebih dari 50% berarti dapat dikatakan bahwa dokumen tersebut mirip.
- e) 100% Hasil uji 100% menandakan bahwa dokumen tersebut adalah sama karena dari awal sampai akhir mempunyai isi yg sama persis.

2.2.1. Algoritma

Algoritma adalah metode efektif yang diekspresikan sebagai rangkaian terbatas. Algoritma juga merupakan kumpulan perintah untuk menyelesaikan suatu masalah. Perintah-perintah ini dapat diterjemahkan secara bertahap dari awal hingga akhir. Masalah tersebut dapat berupa apa saja, dengan syarat untuk setiap permasalahan memiliki kriteria kondisi awal yang harus dipenuhi sebelum menjalankan sebuah algoritma. Algoritma juga memiliki pengulangan proses (iterasi), dan juga memiliki keputusan hingga keputusan selesai (Gun, 2017).

Algoritma berasal dari nama seorang Ilmuwan Arab yang bernama Abu Jafar Muhammad Ibnu Musa Al Khuwarizmi penulis buku berjudul *Al Jabar Wal*

Muqabala. Kata *Al Khuwarizmi* dibaca orang barat menjadi *Algorism* yang kemudian lambat laun menjadi *Algorithm* diserap dalam bahasa Indonesia menjadi Algoritma. Algoritma dapat diartikan urutan penyelesaian masalah yang disusun secara sistematis menggunakan bahasa yang logis untuk memecahkan suatu permasalahan.

Meski demikian terdapat beberapa definisi algoritma yang lain. Di antaranya menurut Rinaldi Munir, algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis. Sedang menurut Kamus Besar Bahasa Indonesia, definisi algoritma adalah urutan logis pengambilan keputusan untuk pemecahan masalah. Menurut tim Gunadarma:1988, algoritma adalah suatu himpunan berhingga dari instruksi-instruksi yang secara jelas memperinci langkah-langkah proses pelaksanaan, dalam pemecahan suatu masalah tertentu, atau suatu kelas masalah tertentu, dengan dituntut pula bahwa himpunan instruksi tersebut dapat dilaksanakan secara mekanik.

Saat menggunakan logika, sebaiknya jangan berfikir terlalu rumit tentang sebuah masalah, karena belum tentu masalah itu serumit yang kita pikir. Pikirkan hal yang paling sederhana untuk menyelesaikan masalah itu, sehingga tidak terjebak dalam pikiran rumit yang dibuat sendiri. Meski demikian jangan meremehkan masalah sekecil apapun, tapi berfikir sederhana untuk menghasilkan solusi yang efektif.

Dalam menentukan algoritma untuk menyelesaikan suatu permasalahan, mungkin kita dihadapkan oleh beberapa pilihan algoritma. Oleh karena itu kita

harus memiliki rambu-rambu dalam menentukan pilihan algoritma. Pertimbangan dalam pemilihan algoritma adalah, pertama, algoritma haruslah benar. Artinya algoritma akan memberikan keluaran sesuai seperti yang diharapkan dari sejumlah masukan yang diberikan. Tidak peduli sebegas apapun algoritma, jika memberikan keluaran yang salah, maka sudah pasti algoritma tersebut bukanlah algoritma yang baik. Pertimbangan kedua yang harus diperhatikan adalah kita harus mengetahui seberapa baik hasil yang dicapai oleh algoritma tersebut. Hal ini penting terutama pada algoritma yang memerlukan aproksimasi hasil yaitu algoritma yang hasilnya hanya berupa pendekatan. Algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai yang sebenarnya. Ketiga adalah efisiensi algoritma. Efisiensi algoritma dapat ditinjau dari dua hal yaitu efisiensi waktu dan memori.

Meskipun algoritma memberikan keluaran yang benar atau paling mendekati, tetapi jika kita harus menunggu lama untuk mendapatkan hasil sampai berjam-jam untuk mendapatkan keluarannya maka biasanya algoritma tersebut biasanya tidak akan menjadi pilihan utama, setiap orang menginginkan keluaran yang relatif cepat. Begitu juga dengan memori, semakin besar memori yang terpakai maka semakin jelek algoritma tersebut. Dalam kenyataannya, setiap orang bisa membuat algoritma yang berbeda untuk menyelesaikan suatu permasalahan, walaupun terjadi perbedaan dalam menyusun algoritma, tentunya kita mengharapkan keluaran yang mirip atau sama. Jika dihadapkan pada permasalahan seperti ini maka sebaiknya pilih algoritma yang paling efisien dan cepat.

Menurut Donald E. Knuth, algoritma mempunyai lima ciri penting yang meliputi:

1. *Finiteness* (keterbatasan), algoritma harus berakhir setelah mengerjakan sejumlah langkah proses.
2. *Definiteness* (kepastian), setiap langkah harus didefinisikan secara tepat dan tidak berarti ganda.
3. *Input* (masukan), algoritma memiliki nol atau lebih data masukan (*input*).
4. *Output* (keluaran), algoritma mempunyai nol atau lebih hasil keluaran (*output*).
5. *Effectiveness* (efektivitas), algoritma harus sangkil (*efektif*), langkah-langkah algoritma dikerjakan dalam waktu yang wajar.

Sedang sifat algoritma adalah:

1. Tidak menggunakan simbol atau sintaks dari suatu bahasa pemrograman tertentu.
2. Tidak tergantung pada suatu bahasa pemrograman tertentu.
3. Notasi-notasinya dapat digunakan untuk seluruh bahasa manapun.
4. Algoritma dapat digunakan untuk merepresentasikan suatu urutan kejadian secara logis dan dapat diterapkan di semua kejadian sehari-hari.

Suatu Algoritma dapat terdiri dari tiga struktur dasar, yaitu runtunan, pemilihan dan pengulangan. Ketiga jenis langkah tersebut membentuk konstruksi suatu algoritma. Berikut adalah penjelasan dari tiga struktur tersebut :

1. Runtunan (*sequence*)

Sebuah runtunan terdiri dari satu atau lebih instruksi. Tiap instruksi dikerjakan secara berurutan sesuai dengan urutan penulisannya, yakni sebuah instruksi dilaksanakan setelah instruksi sebelumnya selesai dikerjakan. Urutan dari instruksi menentukan hasil akhir dari suatu algoritma. Bila urutan penulisan berubah maka mungkin juga hasil akhirnya berubah.

2. Pemilihan (*selection*)

Kadangkala terdapat suatu kejadian yang baru akan dikerjakan jika suatu kondisi tertentu telah terpenuhi. Pemilihan yaitu instruksi yang dikerjakan dengan kondisi tertentu. Kondisi adalah persyaratan yang dapat bernilai benar atau salah. Satu atau beberapa instruksi hanya dilaksanakan apabila kondisi bernilai benar, sebaliknya apabila salah maka instruksi tidak akan dilaksanakan.

3. Pengulangan (*repetition*)

Salah satu kelebihan komputer adalah kemampuannya untuk mengerjakan pekerjaan yang sama berulang kali tanpa mengenal lelah. Kita tidak perlu menulis instruksi yang sama berulang kali, tetapi cukup melakukan pengulangan dengan instruksi yang tersedia. Pengulangan merupakan kegiatan mengerjakan sebuah atau sejumlah aksi yang sama sebanyak jumlah yang ditentukan atau sesuai dengan kondisi yang diinginkan.

2.2.2. Jaro-Winkler

Jaro-Winkler distance merupakan varian dari *Jaro distance metric* yaitu sebuah algoritma untuk mengukur kesamaan antara dua *string*, biasanya algoritma ini digunakan di dalam pendeteksian duplikat. Semakin tinggi *Jaro-Winkler*

distance untuk dua *string*, semakin mirip pula dengan *string* tersebut. Skor normalnya seperti 0 menandakan tidak ada kesamaan, dan 1 adalah sama persis.

Algoritma Jaro-Winkler distance memiliki kompleksitas waktu *quadratic runtime complexity* yang sangat efektif pada *string* pendek dan dapat bekerja lebih cepat dari algoritma *edit distance*.

Langkah dasar untuk menghitung algoritma Jaro antara dua *string* s_1 dan s_2 adalah sebagai berikut (Winkler, 2006) :

1. Menghitung panjang *string* s_1 dan s_2 .
2. Menemukan jumlah karakter yang “sama persis” (m) dalam 2 *string* yang di bandingkan.
3. Menghitung jumlah dari transposisi (t).

Algoritma Jaro mendefinisikan “karakter yang sama” sebagai karakter pada kedua *string* yang sama dan memenuhi ketentuan jarak teoritis. Jarak teoritis dua buah karakter yang disamakan dapat dibenarkan jika melebihi nilai persamaan berikut ini :

$$\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1$$

Pada Algoritma Jaro nilai perbandingan untuk menghitung jarak (d_j) antara dua *string* s_1 dan s_2 dapat di rumuskan dengan :

$$d_j = \frac{1}{3} \times \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right)$$

dimana :

m = jumlah karakter yang sama persis dari dua string yang dibandingkan

$|s_1|$ = panjang string 1

$|s_2|$ = panjang string 2

t = jumlah transposisi

d_j = nilai *Jaro Distance* antara *string 1* dan *string 2*

Akan tetapi bila mengacu kepada nilai yang akan dihasilkan oleh algoritma *Jaro-Winkler* maka nilai jarak maksimalnya adalah 1 yang menandakan kesamaan *string* yang dibandingkan mencapai seratus persen atau sama persis. Biasanya s_1 digunakan sebagai acuan untuk urutan di dalam mencari transposisi. Yang dimaksud transposisi di sini adalah karakter yang sama dari string yang dibandingkan akan tetapi urutannya tertukar. Sebagai contoh, dalam membandingkan kata CRATE dengan TRACE, bila dilihat seksama maka dapat dikatakan semua karakter yang ada di s_1 ada dan sama dengan karakter yang ada di s_2 , tetapi dengan urutan yang berbeda. Dengan mengganti C dan T, dapat dilihat perubahan kata CRATE menjadi TRACE. Pertukaran dua elemen *string* inilah adalah contoh nyata dari transposisi yang dijelaskan. Dalam pencocokan DWAYNE dan DuANE memiliki urutan yang sama D-A-N-E, jadi tidak ada transposisi.

Jaro-Winkler merupakan perpanjangan dari jarak *Jaro*. Ekstensi ini didasarkan pada observasi *Winkler's* bahwa kesalahan pengetikan paling sering terjadi di tengah atau di akhir kata, tetapi sangat jarang di awal. Oleh karena itu, sah

untuk lebih menekankan pada pencocokan prefiks jika jarak Jaro melebihi "ambang batas" yang awalnya ditetapkan ke 0,7 oleh Winkler dalam terbitannya. Jadi apabila nilai d_j dibawah 0.7 maka nilai d_w tidak akan di hitung tetapi apabila nilai d_j di atas 0.7 maka kita akan menghitung nilai dari d_w .

Jaro-Winkler distance menggunakan *prefix scale* (p) yang memberikan tingkat penilaian yang lebih, dan *prefix length* (l) yang menyatakan panjang awalan yaitu panjang karakter yang sama dari string yang dibandingkan sampai ditemukannya ketidaksamaan. Bila *string* s_1 dan s_2 yang diperbandingkan, maka *Jaro-Winkler distancenya* (d_w) adalah:

$$d_w = d_j + (lp(1 - d_j))$$

dimana :

d_j = *Jaro distance* untuk *string* s_1 dan s_2

l = panjang prefiks umum di awal string nilai maksimalnya 4 karakter(panjang karakter yg sama sebelum ditemukan ketidaksamaan max 4)

p = konstanta *scaling factor*. Nilai standar untuk konstanta ini menurut *Winkler* adalah $p = 0,1$

d_w = nilai *Jaro Winkler Distance*

Berikut ini adalah contoh pada perhitungan *JaroWinkler distance*. Jika *string* s_1 MARTHA dan s_2 MARHTA maka:

$m = 6$

$$s_1 = 6$$

$$s_2 = 6$$

Karakter yang tertukar hanyalah T dan H. Maka $t = 1$.

Maka nilai *Jaro distance* adalah:

$$d_j = \frac{1}{3} \times \left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6} \right) = 0.944$$

Kemudian bila diperhatikan susunan s_1 dan s_2 dapat diketahui nilai $l = 3$, dan dengan nilai konstan $p = 0.1$. Maka nilai *Jaro-Winkler distance* adalah:

$$d_w = 0.944 + (3 \times 0.1 (1 - 0.944)) = 0.961$$

Jika *string* s_1 DWAYNE dan s_2 DUANE maka:

$$m = 4$$

$$s_1 = 6$$

$$s_2 = 5$$

$t = 0$, hal ini dikarenakan tidak ada karakter yang sama tapi tertukar urutannya.

Karakter seperti D, A, N, E dianggap dalam urutan yang sama.

Maka nilai *Jaro distance* adalah:

$$d_j = \frac{1}{3} \times \left(\frac{4}{6} + \frac{6}{5} + \frac{4-1}{4} \right) = 0.822$$

Kemudian bila diperhatikan susunan s_1 dan s_2 dapat diketahui nilai $l = 1$, dan dengan nilai konstan $p = 0.1$. Maka nilai *Jaro-Winkler distance* adalah:

$$d_w = 0.822 + (1 \times 0.1 (1 - 0.8.22)) = 0.961.$$

Ada beberapa alasan mengapa algoritma yang digunakan dalam pembuatan sistem ini adalah algoritma Jaro-Winkler dibandingkan dengan algoritma similiaritas lain yaitu:

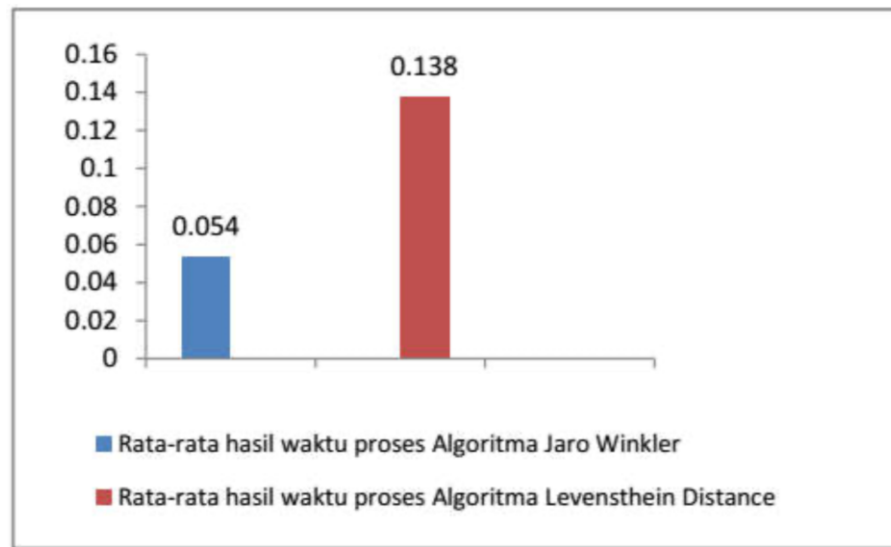
a. Kecepatan Proses Algoritma

Menurut Peter Christen, algoritma Jaro-Winkler memiliki kecepatan pemrosesan data inputan yang lebih cepat dibanding algoritma lain. Pada papernya yang berjudul *A Comparison of Personal Name Matching: Techniques and Practical Issues* (2006), yang membandingkan performa antara algoritma similaritas Jaro Winkler dengan Levenshtein, hasil akhirnya adalah Peter mengurutkan kecepatan algoritma dari yang tercepat sebagai berikut.

1. Jaro
2. Jaro-Winkler
3. Levenshtein
4. Damerau-Levenshtein

Kemudian pada sumber lain yaitu Analisis perbandingan algoritma levenshtein distance dan jaro winkler untuk aplikasi deteksi plagiarisme dokumen teks oleh Michael Julian Tannga, dkk. Yang melakukan penelitian mengenai perbandingan algoritma levenshtein dan jaro winkler, diperoleh kesimpulan bahwa pada data uji yang sama, Algoritma Jaro Winkler memiliki rata-rata waktu proses yang lebih cepat yaitu 0.054 detik, lebih

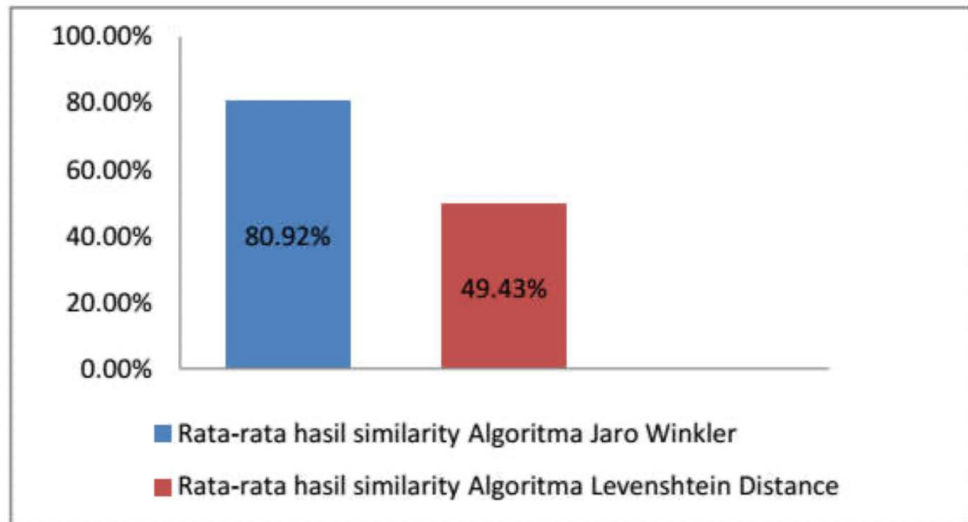
cepat dibandingkan algoritma Levenshtein yaitu 0.138 detik. Data dalam bentuk diagram dapat dilihat pada gambar berikut.



Gambar 2.1 Grafik Rata-rata Kecepatan Proses Algoritma

b. Akurasi Algoritma

Berdasarkan penelitian yang dilakukan oleh Michael Julian Tangga, dkk dengan menggunakan data uji yang sama, perbandingan algoritma Levenshtein dan algoritma Jaro Winkler yang hasilnya dianalisis, kemudian analisis perbandingan yang didapat dan analisis perbandingan yang didapat dari pengujian nilai similarity, yaitu algoritma Jaro Winkler memiliki nilai similarity yang lebih akurat yaitu sebesar 80.92% dibanding dengan algoritma Levenshtein Distance yaitu hanya sebesar 49.43%.



Gambar 2.2 Grafik Rata-rata Hasil Perbandingan Algoritma

2.3. Bahasa Pemrograman

Bahasa pemrograman, atau sering diistilahkan juga dengan bahasa komputer, adalah instruksi standar untuk memerintah komputer. Bahasa pemrograman ini merupakan suatu himpunan dari aturan sintaks dan semantik yang dipakai untuk mendefinisikan program komputer.

Bahasa Pemrograman merupakan prosedur/tata cara penulisan program. Pada bahasa pemrograman terdapat dua faktor penting, yaitu sintaks dan semantik. Sintaks adalah aturan gramatikal yang mengatur tata cara penulisan kata, ekspresi dan pernyataan. Semantik adalah aturan-aturan untuk menyatakan arti. Fungsi Bahasa pemrograman adalah sebagai media untuk menyusun dan memahami serta sebagai alat komunikasi antara pemrogram dengan komputer.

2.3.1. HTML

HTML (*Hyper Text Markup Language*) merupakan bahasa yang digunakan untuk mendeskripsikan struktur sebuah halaman web. HTML berfungsi untuk

mempublikasikan dokumen *online*. *Statement* dasar dari HTML disebut *tag*. Sebuah *tag* dinyatakan dalam sebuah kurung siku (< >). *Tags* yang ditujukan untuk sebuah dokumen atau bagian dari suatu dokumen haruslah dibuat berupa pasangan. Terdiri dari *tag* pembuka dan *tag* penutup. Dimana *tag* penutup menggunakan tambahan tanda garis miring (/) di awal nama *tag* (Henderson, 2009).

HTML yang merupakan singkatan dari *Hyper Text Markup Language* adalah serangkaian kode program yang merupakan dasar dari representasi visual sebuah halaman Web. Didalamnya berisi kumpulan informasi yang disimpan dalam *tag-tag* tertentu, dimana *tag-tag* tersebut digunakan untuk melakukan format terhadap informasi yang dimaksud. Berbagai pengembangan telah dilakukan terhadap kode HTML dan telah melahirkan teknologi-teknologi baru di dalam dunia pemrograman web. Kendati demikian, sampai sekarang HTML tetap berdiri kokoh sebagai dasar dari bahasa web seperti PHP, ASP, JSP dan lainnya. Bahkan secara umum, mayoritas situs web yang ada di Internet pun masih tetap menggunakan HTML sebagai teknologi utama mereka. (Frederick Constantianus, dkk. 2005).

2.3.2. PUG

PUG adalah *template engine* yang diimplementasikan dengan *javascript* untuk *NodeJS* dan *web browser*. Fungsi utama dari PUG adalah untuk memproses *template* dan konten menjadi dokumen HTML.

Berkas *template* PUG (disimpan dengan ekstensi `.pug`) ditulis dengan menggunakan *syntax* yang lebih sederhana dari HTML, sebagai contoh pada gambar 2.3:

```
1 | doctype html
2 | html(lang="id")
3 |   head
4 |     title HTML From Pug
5 |   body
6 |     .container
7 |       h1 Hello World
8 |       p Quick Brown Fox Jumps Over The Lazy Dog
```

Gambar 2.3 Syntax template PUG (disimpan dengan ekstensi `.pug`)

Kode pada gambar 2.3 akan menghasilkan halaman HTML dengan kode seperti pada gambar 2.4 :

```
1 | <!DOCTYPE html>
2 | <html lang="id">
3 |   <head>
4 |     <title>HTML From Pug</title>
5 |   </head>
6 |   <body>
7 |     <div class="container">
8 |       <h1>Hello World</h1>
9 |       <p>Quick Brown Fox Jumps Over The Lazy Dog</p>
10 |     </div>
11 |   </body>
12 | </html>
```

Gambar 2.4 Kode menghasilkan halaman HTML

Program yang digunakan pada gambar 2.3 dan gambar 2.4 adalah *pug-cli*. Program ini menggunakan antarmuka *command line*, yang artinya, program ini digunakan melalui Terminal atau *Command Prompt*. Program ini diinstal menggunakan *package manager* npm yang disediakan dalam *NodeJS*.

2.3.3. CSS

CSS (*Cascading Style Sheet*) adalah salah satu bahasa desain WEB (*style sheet language*) yang mengontrol format tampilan sebuah halaman *web* yang ditulis dengan menggunakan penanda (*markup language*). Biasanya CSS digunakan untuk mendesain sebuah halaman HTML dan XHTML, tetapi sekarang CSS bisa diaplikasikan untuk segala dokumen XML, termasuk SVG dan XUL bahkan ANDROID.

CSS dibuat untuk memisahkan konten utama dengan tampilan dokumen yang meliputi *layout*, warna dari *font*. Pemisahan ini dapat meningkatkan daya akses konten pada web, menyediakan lebih banyak fleksibilitas dan kontrol dalam spesifikasi dari sebuah karakteristik dari sebuah tampilan, memungkinkan untuk membagi halaman untuk sebuah *formatting* dan mengurangi kerumitan dalam penulisan kode dan struktur dari konten, contohnya teknik *tableless* pada desain web.

CSS juga memungkinkan sebuah halaman untuk ditampilkan dalam berbagai *style* dengan menggunakan metode pembawaan yang berbeda pula, seperti *on-screen*, *in-print*, *by voice*, dan lain-lain. Sementara itu, pemilik konten *web* bisa menentukan link yang menghubungkan konten dengan file CSS.

Tujuan utama CSS diciptakan untuk membedakan konten dari dokumen dan dari tampilan dokumen, dengan itu, pembuatan ataupun pemrograman ulang *web* akan lebih mudah dilakukan. Hal yang termasuk dalam desain *web* diantaranya adalah warna, ukuran dan *formatting*. Dengan adanya CSS, konten dan desain *web*

akan mudah dibedakan, jadi memungkinkan untuk melakukan pengulangan pada tampilan-tampilan tertentu dalam suatu *web*, sehingga akan memudahkan dalam membuat halaman *web* yang banyak, yang pada akhirnya dapat memangkas waktu pembuatan *web*.

Fungsi utama CSS adalah merancang, merubah, mendesain, membentuk halaman website. dan isi dari halaman website adalah *tag-tag* html, logikanya CSS itu dapat merubah *tag-tag* html(yang sederhana) sehingga menjadi lebih fungsional dan menarik.

2.3.4. JavaScript

JavaScript adalah bahasa pemrograman *web* yang bersifat *Client Side Programming Language*. *Client Side Programming Language* adalah tipe bahasa pemrograman yang pemrosesannya dilakukan oleh *client*. Aplikasi *client* yang dimaksud merujuk kepada *web browser* seperti *Google Chrome*, *Mozilla Firefox*, *Opera Mini* dan sebagainya.

JavaScript pertama kali dikembangkan pada pertengahan dekade 90'an. Meskipun memiliki nama yang hampir serupa, *JavaScript* berbeda dengan bahasa pemrograman *Java*. Untuk penulisannya, *JavaScript* dapat disisipkan di dalam dokumen HTML ataupun dijadikan dokumen tersendiri yang kemudian diasosiasikan dengan dokumen lain yang dituju. *JavaScript* mengimplementasikan fitur yang dirancang untuk mengendalikan bagaimana sebuah halaman *web* berinteraksi dengan penggunanya (Henderson, 2009).

2.3.4.1. NodeJS

NodeJS adalah perangkat lunak yang didesain untuk mengembangkan aplikasi berbasis *web* dan ditulis dalam sintaks bahasa pemrograman *JavaScript*. Bila selama ini *JavaScript* dikenal sebagai bahasa pemrograman yang berjalan disisi *client/browser*, maka *NodeJS* ada untuk melengkapi peran *JavaScript* sehingga bisa juga berlaku sebagai bahasa pemrograman yang berjalan di sisi *server*, seperti halnya *PHP*, *Ruby*, *Perl*, dan sebagainya. *NodeJS* dapat berjalan di sistem operasi *Windows*, *Mac OS X* dan *Linux* tanpa perlu ada perubahan kode program. *NodeJS* memiliki pustaka *server HTTP* sendiri sehingga memungkinkan untuk menjalankan *server web* tanpa menggunakan program *server web* seperti *Apache* atau *Nginx*.

Untuk mengeksekusi *JavaScript* sebagai bahasa *server* diperlukan *engine* yang cepat dan mempunyai performa yang bagus. *Engine JavaScript* dari *Google* bernama *V8* yang dipakai oleh *NodeJS* agar pengekseskuan *JavaScript* bisa lebih cepat. *Engine* ini juga dipakai oleh *browser Google Chrome*.

Berbeda dengan bahasa pemrograman *server* pada umumnya yang bersifat *blocking*. *NodeJS* bersifat *non-blocking*, sebagaimana halnya *JavaScript* bekerja, *NodeJS* berjalan dengan basis *event(event-driven)*. Maksud dari *blocking* secara sederhana adalah suatu kode program akan dijalankan hingga selesai, kemudian beralih ke kode program selanjutnya. Sedangkan *NodeJS* yang bersifat *non-blocking*, hanya akan mengeksekusi kode saat ada *event* yang diakses.

NodeJS pertama kali diciptakan dan diperkenalkan oleh Ryan Dahl, pada tahun 2009 sehingga *JavaScript* bisa digunakan sebagai bahasa pemrograman di sisi *server*, sekelas dengan PHP, ASP, C#, *Ruby* dan lain sebagainya. Ryan Dahl adalah seorang pengembang dari Joyent yang merupakan sebuah perusahaan perangkat lunak dan infrastruktur *Cloud*. Ia memiliki ketertarikan dengan penerapan *single-threaded* pada bahasa pemrograman sisi *server* dan akhirnya memilih *JavaScript* sebagai bahasa untuk *NodeJS*, setelah sebelumnya mencoba menggunakan *Haskell*, *Lua* dan C.

Kelebihan Memakai *NodeJS* adalah sebagai berikut.:

1. *NodeJS* menggunakan bahasa pemrograman *JavaScript* yang diklaim sebagai bahasa pemrograman yang paling populer dan banyak dikenal oleh masyarakat luas.
2. *NodeJS* mampu menangani ribuan koneksi bersamaan dengan penggunaan *resource* minimum untuk setiap prosesnya.
3. *NodeJS* sangat diandalkan terutama untuk membuat aplikasi *real-time*.
4. *NodeJS* adalah *project open source*, sehingga siapapun dapat melihat struktur kode dan juga dapat berkontribusi untuk pengembangannya.
5. Penggunaan *JavaScript* di sisi *server* dan juga *client* meminimalisir ketidakcocokan antar dua sisi lingkungan pemrograman, seperti terkait komunikasi data yang mana menggunakan struktur JSON yang sama di kedua sisi, validasi *form* yang sama yang dapat dijalankan di sisi *server* dan *client*, dan sebagainya.

6. *Database NoSQL* seperti *MongoDB* dan *CouchDB* mendukung langsung *JavaScript* sehingga *interfacing* dengan *database* ini akan jauh lebih mudah.
7. *NodeJS* memakai *V8* yang selalu mengikuti perkembangan standar *ECMAScript* (nama standar resmi dari *JavaScript*, Namun *JavaScript* yang lebih dikenal dalam implementasinya), sehingga tidak perlu ada kekhawatiran bahwa *browser* tidak mendukung fitur-fitur di *NodeJS*.

2.3.4.2. Esprima

Esprima merupakan sebuah *tool* yang digunakan untuk melakukan *syntactic analysis* maupun *lexical analysis* pada program *JavaScript*. *Esprima* itu sendiri ditulis dengan menggunakan bahasa *JavaScript*.

1. *Environment* yang didukung

Karena *Esprima* ditulis dalam bahasa *Javascript*, maka *Esprima* dapat berjalan pada berbagai *enviromtent* yang biasa digunakan dalam pengembangan *software* berbasis *JavaScript*, beberapa contohnya antara lain:

- *Web browser modern* (*Edge*, *Firefox*, *Chrome*, *Safari*, dll).
- *Web browser* lawas yang sudah mendukung teknologi *JavaScript* (*Internet Explorer* 10 dll).
- *NodeJS* 4 atau yang lebih baru.
- *JavaScript engine* di *Java*, seperti *Rhino* dan *Nashorn*.

2. Fungsi *Esprima*

- *Parsing / Syntactic Analysis*

Fungsi utama dari *Esprima* adalah untuk melakukan penguraian (*parsing*) terhadap kode program *Javascript*. Hal ini juga dikenal dengan sebagai *syntactic analysis*. *Esprima* akan mengambil nilai *string* yang berisi program *JavaScript valid*, kemudian dari kode program tersebut akan dibuat *syntax tree* (pohon *syntax*), sebuah pohon yang teratur yang mendeskripsikan struktur *syntactic* dari program. Dari hasil penguraian ini, pohon *syntax* yang dihasilkan dapat digunakan untuk berbagai macam tujuan, mulai dari untuk transformasi program hingga analisis program tersebut secara *static*.

Antarmuka fungsi dari *parseScript* atau *parseModule* *Esprima* dapat dilihat pada fungsi sebagai berikut.

```
esprima.parseScript(input, config, delegate)
esprima.parseModule(input, config, delegate)
```

Dimana,

- ***Input*** adalah data string yang berisi program *JavaScript* yang akan diuraikan.
- ***Config*** adalah objek yang digunakan untuk mengubah *behaviour* dari *parser (optional)*.
- ***Delegate*** adalah fungsi *callback* yang akan dipanggil pada setiap *node (optional)*

Argumen *input* pada fungsi diatas bersifat wajib, serta tipe datanya harus dalam bentuk *string*, jika tidak maka hasil penguraian program tidak dapat ditentukan. Sementara argumen lain yaitu *config* dan *delegate* bersifat opsional sehingga bisa diabaikan.

Objek data yang diperoleh dari proses *parseScript/parseModel* adalah sebuah pohon *syntax* (*syntax tree*). Sementara untuk format konfigurasi (*config*) dalam dilihat pada tabel berikut.

Tabel 2.1 adalah daftar konfigurasi pada fungsi *parseScript/parseModel*

Tabel 2.1 Daftar Konfigurasi fungsi *parseScript/parseModel*

Nama	Tipe	Default	Deskripsi
<i>jsx</i>	<i>Boolean</i>	<i>false</i>	Support JSX <i>syntax</i>
<i>range</i>	<i>Boolean</i>	<i>false</i>	Beri anotasi pada setiap node dengan lokasi berbasis indeks datanya
<i>loc</i>	<i>Boolean</i>	<i>false</i>	Beri anotasi pada setiap node dengan lokasi berbasis kolom dan 1 barisnya
<i>tolerant</i>	<i>Boolean</i>	<i>false</i>	Toleransi pada <i>error syntax program</i>
<i>tokens</i>	<i>Boolean</i>	<i>false</i>	Ambil setiap nilai token yang ada pada program
<i>comment</i>	<i>Boolean</i>	<i>false</i>	Ambil setiap baris dan kolom komentar

- *Tokenizer / Lexical Analysis*

Tokenizer atau *lexical analysis* dalam *computer science* dikenal sebagai proses konversi urutan karakter (seperti kode program atau halaman *web*) ke dalam urutan *token* (*string* yang diidentifikasi dan maksud dari *string* tersebut).

Tokenizer atau *lexical analysis* pada *Esprima* menggunakan sebuah *string* sebagai *input* dan menghasilkan *array* dari token, yang merepresentasikan kategori-kategori dari data inputan. Proses ini dikenal sebagai *lexical analysis*.

Antarmuka fungsi untuk proses tokenisasi pada *Esprima* adalah:

```
esprima.tokenize(input, config)
```

dimana,

- **Input** yaitu string berisi kode program yang akan ditokenisasi.
- **Config** adalah objek inputan yang digunakan untuk mengatur behaviour dalam proses tokenisasi (*optional*).

Argumen *input* bersifat wajib dan tipenya harus string untuk memastikan proses tokenisasi bekerja dengan baik. Untuk jenis-jenis konfigurasi yang dapat digunakan dalam proses tokenisasi dapat dilihat pada tabel berikut.

Tabel 2.2 adalah daftar konfigurasi pada proses tokenisasi

Tabel 2.2 Daftar Konfigurasi pada proses tokenis

Nama	Tipe	Default	Deskripsi
<i>range</i>	<i>Boolean</i>	<i>false</i>	Beri anotasi pada setiap <i>token</i> dengan lokasi awal dan akhir berbasis nol
<i>loc</i>	<i>Boolean</i>	<i>false</i>	Beri anotasi pada setiap token dengan lokasi berbasis kolom dan barisnya
<i>comment</i>	<i>Boolean</i>	<i>false</i>	Sertakan setiap baris dan blokir komentar di <i>output</i>

Pada gambar 2.3 adalah contoh REPL *NodeJS session* yang menunjukkan contoh penggunaan *tokenizer Esprima*:

```
$ node
> var esprima = require('esprima')
> esprima.tokenize('answer = 42')
[ { type: 'Identifier', value: 'answer' },
  { type: 'Punctuator', value: '=' },
  { type: 'Numeric', value: '42' } ]
```

Gambar 2.5 Contoh Penggunaan Tokenizer Esprima

Pada contoh tersebut, inputan pada *tokenizer* adalah *string* dengan *value* 'answer = 42', dengan *output* berupa *array* berisi *token* untuk tiap-tiap data inputan, dimana 'answer' dikategorikan sebagai *identifier*, '=' dikategorikan sebagai *punctuator*, dan '42' sebagai *numeric*.

2.3.4.3. *ExpressJS*

ExpressJS adalah satu *web framework* paling populer di dunia *NodeJS*. Dokumentasinya yang lengkap dan penggunaannya yang cukup mudah, dapat membuat kita mengembangkan berbagai produk seperti aplikasi web ataupun *RESTful API*. *ExpressJS* pun dapat digunakan menjadi pijakan untuk membangun *web framework* yang lebih kompleks seperti, *Sails.js*, MEAN (*MongoDB*, *ExpressJS*, *AngularJS*, *NodeJS*) dan MERN (*MongoDB*, *ExpressJS*, *ReactJS*, *NodeJS*). *ExpressJS* dibuat oleh TJ Holowaychuk dan sekarang dikelola oleh komunitas.

Beberapa keunggulan yang dimiliki oleh *ExpressJS* antara lain:

1. Dukungan pembuatan *middleware*.

2. Dukungan terhadap berbagai HTTP *verb* seperti POST, GET, PUT, DELETE, OPTION, HEAD, dan lainnya.
3. Sudah terpasang template *engine Jade*.
4. Manajemen file *static* seperti CSS dan *JavaScript*.
5. Sangat bebas untuk dikostumisasi.

2.4. Database

Database secara umum dapat diartikan sebuah tempat penyimpanan data sebagai pengganti dari sistem konvensional yang berupa dokumen file. *Database* didefinisikan kumpulan data yang dihubungkan secara bersama-sama, dan gambaran dari data yang dirancang untuk memenuhi kebutuhan informasi dari suatu organisasi. Berbeda dengan sistem file yang menyimpan 16 data secara terpisah, pada *database* data tersimpan secara terintegrasi. (Sucipto, 2017).

Kegunaan dari *database* sebagai berikut: (Faried, 2003)

1. Salah satu komponen penting dalam sistem informasi, karena merupakan dasar dalam menyediakan informasi.
2. Menentukan kualitas informasi : akurat, tepat pada waktunya dan relevan. Informasi dapat dikatakan bernilai bila manfaatnya lebih efektif dibandingkan dengan biaya mendapatkannya.
3. Mengurangi duplikasi data (*data redundancy*).
4. Hubungan data dapat ditingkatkan (*data relatability*).
5. Mengurangi pemborosan tempat simpanan luar.

2.4.1. Struktur Data

Struktur Data adalah cara penyimpanan, penyusunan dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien. Dengan kata lain struktur data adalah sebuah skema organisasi, seperti *variabel*, *array* dan lain-lain, yang diterapkan pada data sehingga data dapat diinterpretasikan dan sehingga operasi-operasi spesifik dapat dilaksanakan pada data tersebut. Pemakaian struktur data yang tepat didalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana.

Data adalah representasi dari fakta dunia nyata. Fakta atau keterangan tentang kenyataan yang disimpan, direkam atau direpresentasikan dalam bentuk tulisan, suara, gambar, sinyal atau simbol. Data merupakan suatu nilai yang bisa dinyatakan dalam bentuk konstanta atau variabel. Konstanta menyatakan nilai yang tetap, sedangkan variabel menyatakan nilai yang dapat diubah-ubah selama eksekusi berlangsung.

2.4.2. MongoDB

MongoDB merupakan sebuah sistem basis data yang berbasis dokumen dan termasuk sistem basis data yang menganut paham *NoSQL*. *MongoDB* tidak memiliki yang namanya tabel, kolom dan baris. Dalam *MongoDB* yang ada hanyalah koleksi dan dokumen. Dokumen yang terdapat dalam *MongoDB* dapat memiliki atribut yang berbeda dengan dokumen lain walaupun berada dalam satu koleksi.

2.4.2.1. Mongoose

Mongoose adalah sebuah module pada *NodeJS* yang diinstal menggunakan NPM, berfungsi sebagai penghubung antara *NodeJS* dan *database NoSQL MongoDB*. *Mongoose* menyediakan fitur diantaranya, model data aplikasi berbasis *Schema*. Dan juga termasuk *built-in type casting, validation, query building, business logic hooks* dan masih banyak lagi yang menjadi ke andalan *Mongoose*.

2.4.3. NoSQL

Istilah *NoSQL* diciptakan oleh Carlo Strozzi pada tahun 1998 dan mengacu pada *database non-relasional*, pada tahun 2009 Eric Evans memperkenalkan kembali istilah *NoSQL*. Baru-baru ini, istilah ini memiliki makna lain, yaitu "*Not Only SQL*", istilah yang lebih baik dari sebelumnya yang lebih dikenal dengan —anti relasional, *NoSQL* mengakomodasi tanda yang tidak terstruktur, kehadirannya bukan untuk menggantikan SQL namun kedua teknologi ini dapat saling berdampingan. Perbedaan utama kedua *database* ini adalah SQL memiliki skema yang kaku sementara *database NoSQL* menawarkan desain yang fleksibel yang dapat diubah tanpa *downtime* atau gangguan layanan. *NoSQL* juga dirancang untuk menyimpan data yang didistribusikan untuk kebutuhan data dalam skala besar; misalnya *Facebook* memiliki 500 juta pengguna dan *Twitter* terakumulasi *terabyte data*, *database NoSQL* telah memiliki popularitas yang tinggi, sehingga *database* ini diklaim lebih baik dari *database SQL*, *Database NoSQL* dimotivasi oleh kesederhanaan, *horizontal scaling* dan kontrol yang lebih dalam kesediaan data.

NoSQL menjadi solusi dalam penanganan data dalam jumlah besar yang berkembang pesat saat ini. Data ini biasanya non-terstruktur, kompleks dan tidak cocok digunakan dalam model relasional. Contoh data yang bisa kita rasakan adalah data yang berasal dari *smartphone* yang mencatat lokasi *broadcast* setiap saat, video dan kamera bahkan halaman halaman *website* yang berisi banyak informasi serta dokumen.

Basis data *NoSQL* mempunyai karakteristik BASE (*Basically, Available, Soft state and Eventual Consistency*) yang merupakan kebalikan dari ACID pada basis data SQL. Setelah transaksi yang konsisten, keadaan (*state*) yang didapat adalah keadaan sementara (*soft state*) bukan keadaan tetap (*solid state*). Fokus utama dari BASE adalah ketersediaan permanen. Karakteristik berikutnya adalah CAP (*Consistency, Availability dan Partition*) yang mempunyai tiga prinsip utama:

1. Data tersedia pada semua mesin harus sama di semua aspek dan *update* harus dilakukan terhadap semua mesin.
2. Data harus tersedia secara permanen dan harus dapat diakses setiap saat.
3. Pada saat terjadi kegagalan mesin atau kesalahan lainnya, basis data tetap bekerja dengan baik tanpa ada pekerjaan yang berhenti.
4. Klasifikasi model data *NoSQL* terdiri dari: *Key Value Stores, Document Stores, Column Family Stores and Graph Database*.
5. *Key Value Stores* memiliki kesamaan dengan pemetaan data atau direktori dimana data ditangani oleh sebuah kunci unik, dalam pelaksanaannya sangat sederhana yaitu berdasarkan atribut kunci saja.

6. *Document Stores* merupakan data dalam bentuk dokumen JSON, lebih fleksibel. Secara konsep data ini memiliki jenis : JSON, BSON, XML and BLOBs.
7. *Column Family Stores* juga dikenal dengan data yang berorientasi kolom. Media penyimpanan ini terinspirasi oleh *Google Bigtable*. Data disimpan dalam kelompok sel dalam sebuah kolom, dan kolom tersusun dari group kolom. Secara teori jumlahnya tidak terbatas.
8. *Graph Database* efisien dalam penyimpanan data dalam bentuk grafik atau gambar.

2.4.4. Redis

Redis merupakan sumber yang terbuka, Penyimpanan data pada struktur *memory*, digunakan sebagai *database*, lapisan *caching* atau pesan. Kadang *redis* juga disebut sebagai “*leather man of database*”, itu masih sederhana filosofi desain yang fleksibel menjadikan pilihan yang efektif untuk menyelesaikan saat banyak permintaan saat tugas pemrosesan data. *Redis* merupakan untuk membangun dasar pengetahuan tentang makna yang dalam dari konteks *redis* dan teori tentang teknologi. *Redis* berkerja pada RAM karena saat pengelolaan data lebih cepat.

2.5. Hackathon Starter Pack

Hackathon Starter Pack adalah sebuah *boilerplate* untuk aplikasi *web NodeJS*. *Boilerplate* adalah unit penulisan yang dapat digunakan kembali berulang tanpa perubahan. Dengan ekstensi, ide ini terkadang diterapkan pada pemrograman yang dapat digunakan kembali, atau untuk meng-*generate*.

Hackathon Starter Pack seperti *framework* yang bisa digunakan untuk mengembangkan *web* yang dapat memisahkan antara data dan presentasi sehingga memungkinkan pengembangan sebuah *web* dengan cepat dan memudahkan proses pengelolaan *web* tanpa kehilangan fleksibilitas.

2.6. Sublime Text

Sublime Text adalah *text editor* berbasis *Python*, sebuah *text editor* yang elegan, kaya fitur, *cross platform*, mudah dan *simple* yang cukup terkenal di kalangan *developer* (pengembang) dan desainer. *Sublime Text 3* digunakan sebagai *editor* dari bahasa pemrograman PHP dalam melakukan pengelolaan konten di dalam aplikasi *server*. (Diah, 2017).

Sublime Text adalah aplikasi *text editor* yang digunakan untuk membuka *file* apapun namun sejatinya para *programmer* menggunakannya untuk menulis kode. *Sublime Text* mendukung sejumlah bahasa pemrograman diantaranya C, C++, C#, PHP, CSS, HTML, ASP dan banyak lagi.