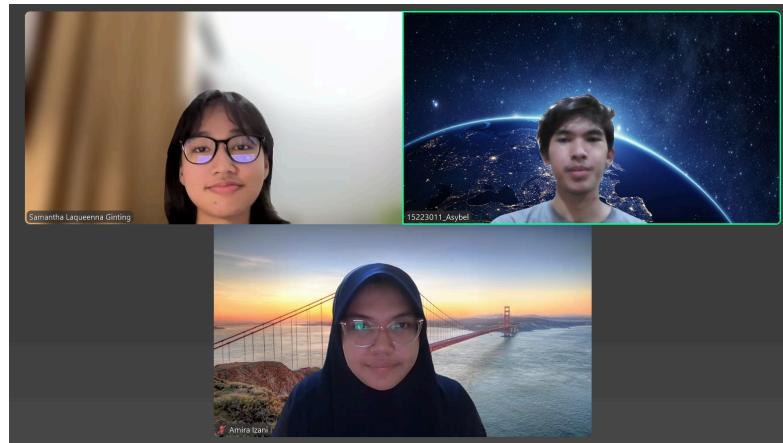


LAPORAN TUGAS BESAR 3 IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2024/2025

PEMANFAATAN PATTERN MATCHING UNTUK MEMBANGUN SISTEM ATS (APPLICANT TRACKING SYSTEM) BERBASIS CV DIGITAL

Kelompok: KakReviewCVAkuDong



Disusun Oleh:

Samantha Laqueenna Ginting (13523138)

Amira Izani (13523143)

Asybel B.P. Sianipar (15223011)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

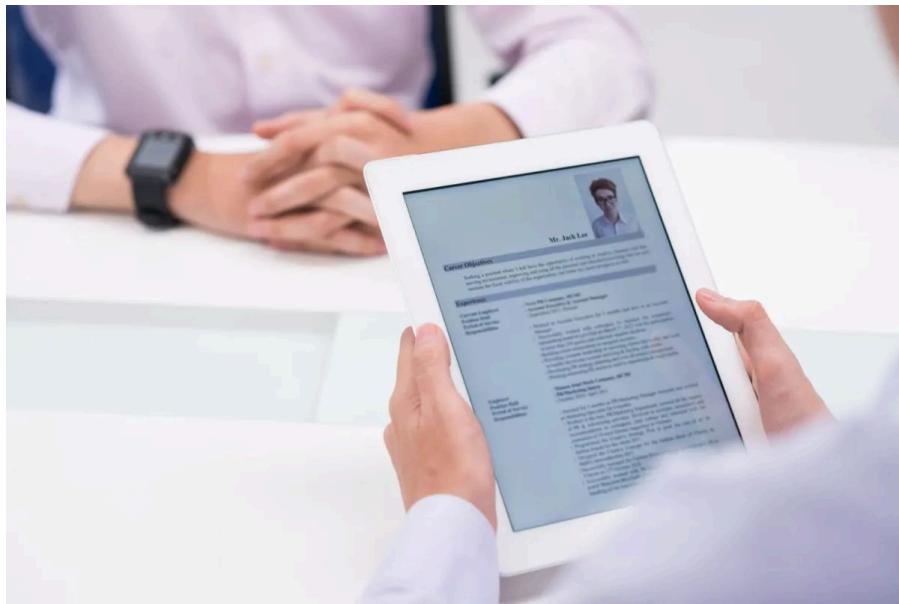
DAFTAR ISI

DAFTAR ISI.....	1
BAB I – DESKRIPSI MASALAH.....	3
Deskripsi Tugas.....	3
Penjelasan Implementasi.....	4
BAB II – LANDASAN TEORI.....	7
2.1. Algoritma Knuth-Morris-Pratt (KMP).....	7
2.2. Algoritma Boyer-Moore (BM).....	9
2.3. Algoritma Aho-Corasick (AC).....	11
Kompleksitas Waktu.....	13
2.4. Fuzzy Matching dan Levenshtein Distance.....	13
2.5. Regular Expression.....	15
BAB III – ANALISIS PEMECAHAN MASALAH.....	16
3.1. Langkah-Langkah Pemecahan Masalah.....	16
3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma KMP dan BM.....	17
3.2.1. Knuth-Morris-Pratt (KMP).....	17
3.2.2. Boyer-Moore (BM).....	19
3.2.3. Aho-Corasick (AC).....	20
3.3. Fitur Fungsional dan Arsitektur Aplikasi yang Dibangun.....	21
3.3.1. Fitur Fungsional.....	21
3.3.2. Arsitektur Aplikasi.....	22
3.4. Contoh Ilustrasi Kasus.....	23
3.4.1. KMP.....	23
3.4.2. Boyer Moore.....	24
3.4.3. Aho-Corasick.....	25
BAB IV – IMPLEMENTASI DAN PENGUJIAN.....	29
4.1. Spesifikasi Teknis Program.....	29
4.2. Penjelasan Tata Cara Penggunaan Program.....	51
Setup Program:.....	51
4.3. Hasil dan Analisis Pengujian Parameter Algoritma.....	54
4.3.1. Pengujian Parameter.....	54
4.3.1. Analisis Hasil Pengujian Parameter.....	55
4.4. Hasil dan Analisis Pengujian Aplikasi ATS.....	56
4.5. Implementasi Bonus.....	60
4.5.1. Aho-Corasick.....	60
4.5.2. Enkripsi.....	61
BAB V – KESIMPULAN, SARAN, DAN REFLEKSI.....	63
5.1. Kesimpulan.....	63

5.2. Saran.....	63
5.3. Refleksi.....	64
DAFTAR PUSTAKA.....	65
LAMPIRAN.....	66
A. Pranala Repository.....	66
B. Pranala Video.....	66
C. Tabel Ketercapaian.....	66

BAB I – DESKRIPSI MASALAH

Deskripsi Tugas



Gambar 1. CV ATS dalam Dunia Kerja

(Sumber: <https://www.antaranews.com/>)

Di era digital ini, keamanan data dan akses menjadi semakin penting. Perkembangan proses rekrutmen tenaga kerja telah mengalami perubahan signifikan dengan memanfaatkan teknologi untuk meningkatkan efisiensi dan akurasi. Salah satu inovasi yang menjadi solusi utama adalah Applicant Tracking System (ATS), yang dirancang untuk mempermudah perusahaan dalam menyaring dan mencocokkan informasi kandidat dari berkas lamaran, khususnya Curriculum Vitae (CV). ATS memungkinkan perusahaan untuk mengelola ribuan dokumen lamaran secara otomatis dan memastikan kandidat yang relevan dapat ditemukan dengan cepat.

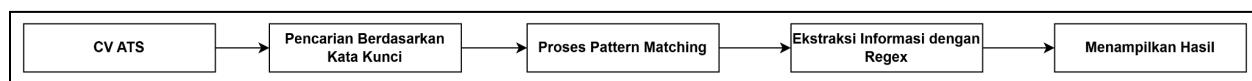
Meskipun demikian, salah satu tantangan besar dalam pengembangan sistem ATS adalah kemampuan untuk memproses dokumen CV dalam format PDF yang tidak selalu terstruktur. Dokumen seperti ini memerlukan metode canggih untuk mengekstrak informasi penting seperti identitas, pengalaman kerja, keahlian, dan riwayat pendidikan secara efisien. Pattern matching menjadi solusi ideal dalam menghadapi tantangan ini.

Pattern matching adalah teknik untuk menemukan dan mencocokkan pola tertentu dalam teks. Dalam konteks ini, algoritma Boyer-Moore dan Knuth-Morris-Pratt (KMP) sering digunakan karena keduanya menawarkan efisiensi tinggi untuk pencarian teks di dokumen besar. Algoritma ini memungkinkan sistem ATS untuk mengidentifikasi informasi penting dari CV pelamar dengan kecepatan dan akurasi yang optimal.

Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan deteksi informasi pelamar berbasis dokumen CV digital. Metode yang akan digunakan untuk melakukan deteksi pola dalam CV adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas kandidat melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali profil pelamar secara lengkap hanya dengan menggunakan CV digital.

Penjelasan Implementasi

Dalam tugas ini, Anda akan mengembangkan sebuah sistem ATS (Applicant Tracking System) berbasis CV Digital dengan memanfaatkan teknik Pattern Matching. Implementasi sistem ini akan menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt (*Aho-Corasick* apabila mengerjakan bonus) untuk menganalisis dan mencocokkan pola dalam dokumen CV digital, sesuai dengan konsep yang telah dipelajari dalam materi dan slide perkuliahan.



Gambar 2. Skema Implementasi *Applicant Tracking System*

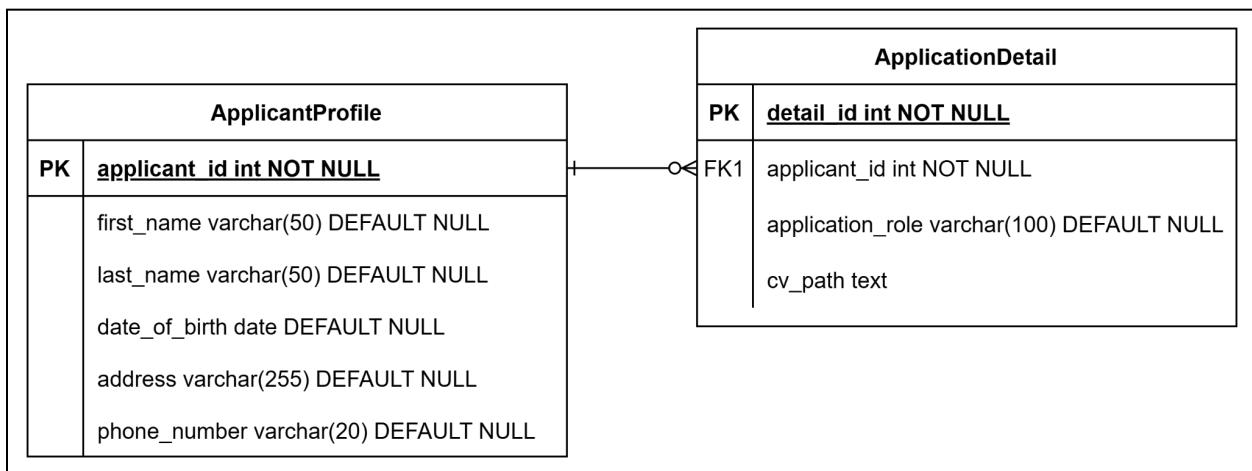
Sistem ini bertujuan untuk mencocokkan kata kunci dari user terhadap isi CV pelamar kerja dengan pendekatan pattern matching menggunakan algoritma KMP (Knuth-Morris-Pratt) atau BM (Boyer-Moore). Semua proses dilakukan secara in-memory, tanpa menyimpan hasil pencarian—hanya data mentah (raw) CV yang disimpan. Pengguna (HR atau rekruter) akan memberikan input berupa daftar kata kunci yang ingin dicari (misalnya: "python", "react", dan "sql") serta jumlah CV yang ingin ditampilkan (misalnya Top 10 matches). Setiap file CV dalam format PDF akan dikonversi menjadi satu string panjang yang memuat seluruh teks dari dokumen tersebut. Proses konversi ini bertujuan untuk mempermudah pencocokan pola menggunakan algoritma string matching, sehingga setiap keyword dapat dicari secara efisien dalam satu representasi data linear.

Untuk memberikan pemahaman yang lebih konkret, berikut disajikan contoh kasus penerapan sistem CV ATS beserta prosesnya dan contoh output yang dihasilkan. Dataset yang digunakan dalam contoh ini merupakan dataset CV ATS yang tercantum pada bagian referensi.

Tabel 1. Hasil ekstraksi teks dari CV ATS

CV ATS	Ekstraksi Text untuk Regex	Ekstraksi Text untuk <i>Pattern Matching</i> (KMP & BM)
 10276858.pdf	Ekstraksi Text Regex.txt	Ekstraksi Text Pattern Matching.txt

Pada tahap implementasi ini, setiap CV yang telah dikonversi menjadi string panjang untuk mempermudah proses pencocokan. Representasi ini menjadi dasar dalam mencari CV yang paling relevan dengan kata kunci yang dimasukkan oleh pengguna. Proses pencarian dilakukan dengan menggunakan algoritma pencocokan string Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) untuk menemukan CV yang memiliki kemiripan tertinggi dengan kebutuhan yang ditentukan. Apabila tidak ditemukan satupun CV dalam basis data yang memiliki kecocokan kata kunci secara exact match menggunakan algoritma KMP maupun Boyer-Moore, maka sistem akan mencari CV yang paling mirip berdasarkan tingkat kemiripan di atas ambang batas tertentu (threshold). Hal ini mempertimbangkan kemungkinan adanya kesalahan pengetikan (typo) oleh pengguna atau HR saat memasukkan kata kunci. Anda diberikan **keleluasaan untuk menentukan nilai ambang batas persentase** kemiripan tersebut, dengan syarat dilakukan pengujian terlebih dahulu untuk menemukan nilai tuning yang optimal dan **dijelaskan secara rinci dalam laporan**. Metode perhitungan tingkat kemiripan harus diterapkan menggunakan algoritma **Levenshtein Distance**.



Gambar 3. Skema basis data CV ATS

Dalam skema basis data ini, tabel **ApplicantProfile** menyimpan informasi pribadi pelamar, sedangkan tabel **ApplicationDetail** menyimpan detail aplikasi yang diajukan oleh pelamar tersebut. Relasi antara tabel **ApplicantProfile** dan **ApplicationDetail** adalah one-to-many, karena seorang pelamar dapat mengajukan lamaran untuk beberapa posisi dalam perusahaan yang sama, atau bahkan perusahaan yang berbeda. Setiap lamaran mungkin memerlukan dokumen yang berbeda, seperti CV yang telah disesuaikan untuk peran tertentu.

Untuk keperluan pengembangan awal, basis data silahkan **di-seeding secara mandiri** menggunakan data simulasi. Mendekati tenggat waktu pengumpulan tugas, asisten akan menyediakan seeding resmi yang akan digunakan untuk Demo Tugas Besar.

Atribut **cv_path** pada tabel **ApplicationDetail** digunakan untuk menyimpan lokasi berkas CV digital pelamar di dalam repositori sistem. Lokasi penyimpanan mengikuti struktur folder di direktori **data/**, sebagaimana dijelaskan dalam struktur *repository* pada bagian [pengumpulan tugas](#). Berkas CV yang tersimpan akan dianalisis oleh sistem ATS (Applicant Tracking System) yang dikembangkan dalam Tugas Besar ini.

BAB II – LANDASAN TEORI

Algoritma Pattern Matching adalah teknik pencocokan pola karakter-karakter tertentu dengan data atau teks yang lebih besar. Teknik ini digunakan untuk mencari kemiripan antara pola yang dicari dengan data yang tersedia, yang sering digunakan dalam pemrosesan teks, analisis data, dan bidang lainnya.

2.1. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) mencari pola dalam data teks dari urutan kiri ke kanan, seperti halnya algoritma brute force dalam pencocokan pola (pattern). Namun, algoritma ini terbukti lebih efisien dari algoritma brute force. KMP akan menghindari pemeriksaan ulang karakter yang telah dibandingkan dengan memanfaatkan informasi sebelumnya yang diperoleh dari pola itu sendiri.

Sebelum melakukan penyelesaian dengan KMP, proses preprocessing dilakukan pada pola dengan membangun sebuah fungsi yang disebut fungsi pinggiran (border function). Fungsi ini merepresentasikan panjang prefiks terpanjang dari pattern yang juga merupakan sufiks dari pattern itu sendiri hingga posisi tertentu. Dengan adanya fungsi ini, ketika terjadi mismatch pada karakter ke- j , algoritma tidak perlu kembali ke posisi awal dalam teks, namun cukup menggeser pola sejauh yang ditentukan oleh nilai fungsi tersebut.

Proses pencocokan dimulai dari kiri ke kanan antara data teks dan pattern. Ketika karakter pada posisi j dalam pattern tidak cocok dengan karakter pada posisi i dalam teks, maka nilai j diset ulang ke nilai $b[j - i]$, dimana $b[]$ adalah array fungsi pinggiran. Jika kemudian nilai j sudah nol, maka indeks i pada teks akan dinaikkan satu. Oleh karena itu, kompleksitas waktu dari algoritma KMP adalah $O(n + m)$ dengan n adalah panjang teks dan m adalah panjang pattern.

Berikut merupakan contoh penggunaannya dalam kode program Java.

```
public static int kmpMatch(String text, String pattern) {  
    int n = text.length();  
    int m = pattern.length();  
  
    int b[] = computeBorder(pattern);  
  
    int i = 0;  
    int j = 0;
```

```

while (i < n) {
    if (pattern.charAt(j) == text.charAt(i)) {
        if (j == m - 1) return i - m + 1; // match
        i++;
        j++;
    }
    else if (j > 0) j = b[j - 1];
    else i++;
}
return -1; // no match
} // end of kmpMatch()

public static int[] computeBorder(String pattern) {
    int b[] = new int[pattern.length()];
    fail[0] = 0;

    int m = pattern.length();
    int j = 0;
    int i = 1;

    while (i < m) {
        if (pattern.charAt(j) == pattern.charAt(i)) {
            // j + 1 chars match
            b[i] = j + 1;
            i++;
            j++;
        }
        else if (j > 0) j = b[j - 1]; // j follows matching prefix
        else { // no match
            b[i] = 0;
            i++;
        }
    }
    return fail;
} // end of computeBorder()

```

2.2. Algoritma Boyer-Moore (BM)

Untuk data teks dan variasi karakter yang lebih besar, algoritma Boyer-Moore sering digunakan untuk pencocokan string. Algoritma ini berbeda dari algoritma KMP karena pencocokan dilakukan dari kanan pattern ke kiri. Keunggulan utama dari algoritma ini terletak pada kemampuannya untuk melakukan lompatan besar dalam teks ketika terjadi mismatch.

Proses preprocessing dilakukan pada pola untuk membangun fungsi last occurrence ($L(x)$), yang menyimpan posisi indeks terakhir dari setiap karakter dalam pola. Ketika mismatch terjadi pada karakter x di teks, algoritma akan mencari apakah karakter x terdapat di dalam pola. Jika ya, pola akan digeser sehingga karakter x dalam teks sejajar dengan kemunculan terakhir x dalam pola. Jika tidak, pola akan digeser sepenuhnya melewati x , menciptakan lompatan yang besar.

Dalam kasus terbaik (best case), kompleksitas dari algoritma BM sublinear, yang jauh lebih cepat dari $O(n)$, karena tidak perlu membandingkan semua karakter teks. Namun pada kasus terburuk (worst case), kompleksitasnya bisa mencapai $O(n \times m)$, seperti halnya kompleksitas algoritma brute force.

Berikut merupakan contoh penggunaannya dalam kode program Java.

```
public static int bmMatch(String text, String pattern) {  
    int last[] = buildLast(pattern);  
    int n = text.length();  
    int m = pattern.length();  
    int i = m - 1;  
  
    if (i > n - 1) return -1; // no match if pattern is  
                           // longer than text  
  
    int j = m - 1;
```

```

do {

    if (pattern.charAt(j) == text.charAt(i)) {

        if (j == 0) return i;

        else { // looking-glass technique

            i--;
            j--;

        }

    }

    else { // character jump technique

        int lo = last[text.charAt(i)]; // last occ

        i = i + m - Math.min(j, 1 + lo);

        j = m - 1;

    }

} while (i <= n - 1);

return -1; // no match
} // end of bmMatch()

public static int[] buildLast(String pattern) {

/* Return array sorting index of last occurrence of each ASCII
char in pattern */

int last[] = new int[128]; // ASCII char set

```

```

for (int i=0; i < 128; i++) {

    last[i] = -1; // init array

}

for (int i=0; i < pattern.length(); i++) {

    last[pattern.charAt(i)] = i;

}

return last;

} // end of buildLast()

```

2.3. Algoritma Aho-Corasick (AC)

Algoritma Aho-Corasick (AC) adalah algoritma pencocokan string yang dirancang untuk mencari semua kemunculan dari sekumpulan pola (biasanya disebut sebagai kamus) dalam sebuah teks secara efisien. Algoritma ini diperkenalkan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975 dan dikenal luas karena kemampuannya mencocokkan banyak pola sekaligus dalam satu kali lintasan teks. Algoritma ini sering digunakan dalam berbagai aplikasi, seperti pencarian teks, deteksi virus, pemrosesan bahasa alami, dan analisis dokumen seperti resume untuk mencari kata kunci tertentu.

Algoritma Aho-Corasick mengintegrasikan konsep mesin keadaan terbatas (finite state machine) dengan struktur data trie (prefix tree). Dengan membangun sebuah automata, algoritma ini memungkinkan pencocokan simultan dari seluruh pola dalam kamus terhadap teks input. Prosesnya terdiri dari dua tahap utama: pembangunan automata dan pencarian pola dalam teks.

Algoritma ini bergantung pada beberapa **komponen kunci**:

1. **Trie (Prefix Tree):**

- Trie dibangun dari kumpulan pola yang akan dicari.
- Setiap node dalam trie merepresentasikan prefiks dari satu atau lebih pola.

- Setiap tepi (edge) dalam trie merepresentasikan sebuah karakter dari pola.

2. Fungsi Transisi (Goto Function):

- Menentukan keadaan berikutnya berdasarkan karakter yang dibaca dari teks.
- Jika tidak ada transisi yang sesuai untuk karakter tertentu, algoritma beralih ke fungsi kegagalan.

3. Fungsi Kegagalan (Failure Function):

- Menangani kasus ketika transisi tidak ditemukan.
- Mengarahkan ke keadaan yang mewakili sufiks terpanjang dari prefiks yang telah dibaca, yang juga merupakan prefiks dari pola lain dalam kamus.

4. Fungsi Output:

- Menyimpan informasi tentang pola yang ditemukan pada setiap keadaan tertentu.

Pembangunan automata dilakukan dalam beberapa langkah:

1. Membangun Trie:

- Semua pola dimasukkan ke dalam trie.
- Setiap jalur dari akar (root) ke daun (leaf) merepresentasikan sebuah pola lengkap.

2. Menghitung Fungsi Kegagalan:

- Menggunakan algoritma pencarian lebar (Breadth-First Search, BFS) untuk mengunjungi setiap node dalam trie.
- Untuk setiap node, fungsi kegagalan dihitung berdasarkan sufiks terpanjang dari prefiks yang telah dibaca yang juga merupakan prefiks dari pola lain.

3. Menghitung Fungsi Output:

- Pada setiap node, simpan daftar pola yang berakhir di node tersebut.
- Sertakan juga pola yang dapat dicapai melalui fungsi kegagalan untuk memastikan semua kemunculan pola terekam.

Proses pencarian dalam teks dilakukan sebagai berikut:

1. Inisialisasi:

- Mulai dari akar (root) trie.

2. Pemrosesan Teks:

- Untuk setiap karakter dalam teks, gunakan fungsi transisi untuk berpindah ke keadaan berikutnya.
- Jika tidak ada transisi yang cocok, gunakan fungsi kegagalan untuk mencari keadaan alternatif.

- Setiap kali mencapai keadaan dengan output, catat pola yang ditemukan pada posisi tersebut.

Kompleksitas Waktu

- **Pembangunan Automata:**
 - Kompleksitasnya adalah $O(m + k)$, di mana m adalah total panjang semua pola dalam kamus dan k adalah ukuran alfabet yang digunakan.
- **Pencarian:**
 - Kompleksitasnya adalah $O(n + z)$, di mana n adalah panjang teks dan z adalah total jumlah kemunculan pola dalam teks.

Algoritma ini sangat efisien untuk kasus dengan banyak pola dan teks yang panjang, karena waktu pencarian bersifat linear terhadap panjang teks ditambah jumlah pola yang ditemukan.

2.4. Fuzzy Matching dan Levenshtein Distance

Fuzzy Matching adalah teknik pencocokan string yang memungkinkan pencocokan meski terdapat perbedaan atau kesalahan kecil dalam teks yang dibandingkan. Algoritma ini bertujuan untuk menemukan kesamaan relatif antar string, sehingga berguna ketika data yang diproses bersifat tidak konsisten, mengandung typo, atau ditulis dalam gaya bebas.

Levenshtein Distance atau edit distance adalah sebuah metrik yang digunakan dalam fuzzy matching. Metrik ini mengukur seberapa banyak operasi pengeditan minimum yang diperlukan untuk mengubah satu string menjadi string lainnya. Operasi yang diperbolehkan meliputi penyisipan (insertion), penghapusan (deletion) dan penggantian (substitution) satu karakter. Contoh, jarak antara string “kitten” dan “sitting” adalah 3, karena diperlukan tiga buah operasi: ‘k’ → ‘s’ (substitusi), ‘e’ → ‘i’ (substitusi), dan menambahkan ‘g’ di akhir (insertion).

Algoritma Levenshtein Distance diimplementasikan menggunakan program dinamis, dengan membangun matriks $m \times n$ (dengan m dan n adalah panjang dua string yang dibandingkan) untuk menyimpan jarak minimum antara semua pasangan substring. Kompleksitas waktu dari algoritma ini adalah $O(m \times n)$.

Berikut merupakan contoh penggunaannya dalam kode program Java.

```
public static int levenshteinDistance(String s1, String s2) {  
    int m = s1.length();  
    int n = s2.length();  
  
    int[][] dp = new int[m + 1][n + 1];  
  
    // Init row and col  
    for (int i = 0; i <= m; i++) dp[i][0] = i;  
    for (int j = 0; j <= n; j++) dp[0][j] = j;  
  
    // Iterating matrix content  
    for (int i = 1; i <= m; i++) {  
        char c1 = s1.charAt(i - 1);  
        for (int j = 1; j <= n; j++) {  
            char c2 = s2.charAt(j - 1);  
            int cost = (c1 == c2) ? 0 : 1;  
  
            dp[i][j] = Math.min(  
                Math.min(dp[i - 1][j] + 1, // deletion  
                    dp[i][j - 1] + 1), // insertion  
                dp[i - 1][j - 1] + cost); // substitution  
        }  
    }  
}
```

```
    return dp[m][n];  
}
```

2.5. Regular Expression

Regular Expression (Regex) adalah teknik yang digunakan untuk mencocokkan dan memproses string secara fleksibel dan efisien. Regex digunakan secara luas dalam berbagai aplikasi seperti editor teks, mesin pencarian web, dan analisis data. Teknik ini bekerja dengan membangun mesin keadaan hingga deterministik (DFA) atau nondeterministik (NFA) yang memproses teks untuk mencari pola yang sesuai dengan regex. Kompleksitas waktunya bergantung pada implementasi dari mesin keadaan, panjang, serta kerumitan regex. Terkadang, regex dapat menjadi kurang efisien untuk pola yang sangat kompleks atau data yang sangat panjang.

BAB III – ANALISIS PEMECAHAN MASALAH

3.1. Langkah-Langkah Pemecahan Masalah.

Pemecahan masalah untuk membangun sistem Applicant Tracking System (ATS) ini dilakukan dengan membagi alur kerja menjadi beberapa tahapan pemrosesan yang terdefinisi dengan jelas. Sistem ini dirancang untuk memproses berkas CV dalam format PDF, mencocokkannya dengan kata kunci yang diberikan oleh pengguna (perekut), dan memberikan peringkat pelamar yang paling relevan. Secara rinci, berikut adalah langkah-langkah pemecahan masalah yang diimplementasikan pada program ini.

1. Pengambilan dan Pra-pemrosesan Data CV

Langkah pertama adalah memastikan data pelamar dikelola dengan aman dan efisien. Sistem akan mengambil data seluruh aplikasi dari basis data MySQL, yang mencakup informasi profil pelamar dan lokasi (*path*) dari berkas CV mereka. Untuk menjaga kerahasiaan dan keamanan data pribadi pelamar, seluruh informasi sensitif (seperti nama, tanggal lahir, kontak, dan alamat) dienkripsi menggunakan algoritma *Vigenère cipher* sebelum disimpan ke dalam basis data. Proses ini memastikan bahwa data tetap aman bahkan jika terjadi akses langsung ke basis data.

Selanjutnya, untuk setiap berkas CV PDF, sistem akan membaca dan mengekstraksi seluruh konten teksnya menjadi sebuah string tunggal yang panjang. Teks ini kemudian diubah menjadi huruf kecil (*lowercase*) untuk memastikan proses pencocokan tidak sensitif terhadap besar kecilnya huruf (*case-insensitive*). Saat data profil ditampilkan kepada pengguna, data tersebut akan didekripsi secara transparan sehingga pengguna tetap melihat informasi yang dapat dibaca.

2. Pencocokan kata kunci tepat (*exact matching*)

Sistem melakukan iterasi pada setiap teks CV yang telah diekstraksi. Untuk setiap CV, sistem mencari semua kata kunci yang diberikan pengguna menggunakan algoritma yang telah dipilih (KMP, Boyer-Moore, Aho-Corasick). Jika sebuah kata kunci ditemukan, sistem akan mencatat CV pelamar tersebut beserta kata kunci yang cocok dan frekuensi kemunculannya. Hasil ini disimpan sementara untuk diproses pada tahap selanjutnya.

3. Pencocokan perkiraan (*fuzzy matching*)

Setelah fase *exact matching* selesai, sistem mengidentifikasi kata kunci mana yang tidak ditemukan di dalam sebuah CV. Untuk setiap kata kunci yang tidak ditemukan tersebut, sistem akan melakukan *fuzzy matching*. Algoritma Levenshtein Distance digunakan untuk menghitung kemiripan antara kata kunci tersebut dengan setiap kata yang ada di dalam teks CV. Jika tingkat kemiripan (similarity score) melebihi ambang batas yang telah ditentukan (misalnya, 80%), kata tersebut dianggap sebagai "fuzzy match". Sistem akan menyimpan kecocokan ini beserta skor kemiripannya. Langkah ini penting untuk mengatasi kesalahan pengetikan (*typo*) atau variasi kata.

4. Penilaian dan Pemeringkatan

Setiap CV yang memiliki setidaknya satu kecocokan (baik *exact* maupun *fuzzy*) akan diberi skor. Skor dihitung berdasarkan jumlah kata kunci unik yang cocok, di mana setiap *exact match* memberikan bobot penuh (1.0) dan setiap *fuzzy match* memberikan bobot sesuai skor kemiripannya. Sebagai *tie-breaker*, ditambahkan bonus kecil berdasarkan total frekuensi kemunculan kata kunci *exact match*. Seluruh CV yang telah dinilai kemudian diurutkan dari skor tertinggi ke terendah. Sistem akan mengambil sejumlah CV teratas sesuai dengan input "Top-N" dari pengguna.

5. Ekstraksi detail dan tampilan hasil

Hasil peringkat ditampilkan kepada pengguna dalam bentuk kartu-kartu ringkasan. Setiap kartu berisi nama pelamar, daftar kata kunci yang cocok, dan tombol untuk melihat detail lebih lanjut.

Jika pengguna memilih untuk melihat ringkasan detail (*summary*) dari seorang pelamar, sistem akan menggunakan *Regular Expression* (Regex) untuk mengekstrak informasi terstruktur dari teks mentah CV, seperti keahlian (*skills*), pengalaman kerja, dan riwayat pendidikan, lalu menampilkannya di halaman khusus.

3.2. Proses Pemetaan Masalah Menjadi Elemen-Elemen Algoritma KMP dan BM.

3.2.1. Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) digunakan untuk pencocokan string secara efisien dengan memanfaatkan kegagalan dalam pencocokan sebelumnya untuk menghindari perbandingan yang berulang.

- **Pemetaan Elemen** Kunci dari algoritma KMP adalah tahap pra-pemrosesan pada pola (kata kunci) untuk membuat sebuah *array* bantu yang disebut tabel *Longest Proper Prefix which is also Suffix* (LPS). Tabel ini menyimpan panjang dari prefiks sejati terpanjang dari sebuah sub-pola yang juga merupakan sufiks dari sub-pola tersebut.
 1. **Inisialisasi:** Sebuah array lps dengan ukuran yang sama dengan panjang kata kunci dibuat. $lps[0]$ selalu diatur ke 0.
 2. **Konstruksi Tabel:** Algoritma melakukan iterasi melalui kata kunci. Untuk setiap posisi i , ia menemukan panjang prefiks terpanjang yang juga merupakan sufiks dari $pattern[0...i]$. Jika $pattern[i]$ cocok dengan $pattern[length]$, nilai $length$ akan bertambah dan $lps[i]$ diisi dengan nilai $length$ yang baru. Jika tidak cocok, $length$ akan diperbarui ke nilai dari $lps[length-1]$ untuk mencoba mencocokkan dengan prefiks yang lebih pendek.
- **Penyelesaian Solusi** Setelah tabel LPS dibuat, proses pencocokan pada teks CV dilakukan sebagai berikut:
 1. **Iterasi Melalui Teks:** Dua penunjuk (indeks), i untuk teks CV dan j untuk kata kunci, digunakan.
 2. **Pengecekan Kecocokan:**
 - Jika karakter $text[i]$ dan $pattern[j]$ cocok, kedua indeks i dan j dinaikkan.
 - Jika j mencapai panjang kata kunci, artinya satu kecocokan penuh telah ditemukan. Indeks awal kecocokan ($i-j$) dicatat, dan j diatur ulang menggunakan tabel LPS ($j = lps[j-1]$) untuk melanjutkan pencarian sisa kemunculan di teks.
 3. **Penanganan Ketidakcocokan (Mismatch):**
 - Jika $text[i]$ dan $pattern[j]$ tidak cocok:
 - Jika j tidak sama dengan 0, maka j diatur ulang ke $lps[j-1]$. Ini adalah langkah kunci KMP, di mana program tidak menggeser penunjuk i pada teks, melainkan hanya menggeser penunjuk j pada pola ke posisi yang tepat untuk melanjutkan perbandingan.

- Jika j sama dengan 0, artinya karakter pertama pola sudah tidak cocok, maka hanya indeks i pada teks yang digeser ke kanan.

3.2.2. Boyer-Moore (BM)

Algoritma Boyer-Moore (BM) seringkali lebih cepat dari KMP dalam praktik karena ia membandingkan pola dari kanan ke kiri dan menggunakan heuristik untuk melakukan lompatan besar pada teks.

- **Pemetaan Elemen** Dalam implementasi ini, kami menggunakan heuristik *Last Occurrence* (atau dikenal sebagai *Last Occurrence Table*). Sebuah tabel dibuat pada tahap pra-pemrosesan untuk menyimpan informasi tentang posisi kemunculan terakhir setiap karakter yang mungkin ada di dalam kata kunci.
 1. **Inisialisasi Tabel:** Sebuah tabel (misalnya, *array* atau *hash map*) diinisialisasi untuk semua karakter dalam set karakter yang digunakan (misalnya, 256 untuk ASCII). Setiap entri diinisialisasi dengan nilai -1 (atau nilai lain yang menandakan karakter tidak ada dalam pola).
 2. **Pengisian Tabel:** Algoritma melakukan iterasi melalui kata kunci. Untuk setiap karakter pada $\text{pattern}[i]$, entri tabel yang sesuai untuk karakter tersebut diatur ke i .
- **Penyelesaian Solusi**
 1. **Penjajaran Awal:** Pola disejajarkan dengan awal teks.
 2. **Perbandingan dari Kanan ke Kiri:** Proses perbandingan dimulai dari karakter terakhir pola.
 3. **Penanganan Kecocokan:** Jika karakter cocok, perbandingan bergerak satu langkah ke kiri pada teks dan pola. Jika seluruh karakter pola cocok, sebuah kemunculan ditemukan.
 4. **Penanganan Ketidakcocokan (*Mismatch*):** Jika terjadi ketidakcocokan pada $\text{text}[i]$ saat dibandingkan dengan $\text{pattern}[j]$, algoritma melakukan pergeseran berdasarkan tabel *Last Occurrence*:
 - Nilai pergeseran dihitung untuk menggeser pola ke kanan sehingga karakter $\text{text}[i]$ sejajar dengan kemunculan terakhirnya di dalam pola.
 - Pola digeser sejauh nilai yang telah dihitung, yang seringkali lebih dari satu posisi, dan proses perbandingan diulang.

3.2.3. Aho-Corasick (AC)

Algoritma Aho-Corasick (AC) dirancang khusus untuk mencari semua kemunculan dari sekumpulan kata kunci (kamus) dalam sebuah teks secara bersamaan dalam satu kali proses. Efisiensinya yang tinggi menjadikannya pilihan ideal untuk kasus ATS di mana perekut memasukkan banyak kata kunci sekaligus.

Tahap pra-pemrosesan pada Aho-Corasick adalah membangun sebuah mesin status hingga (finite automaton) dari semua kata kunci yang diberikan. Proses ini menggabungkan struktur data Trie dengan konsep dari algoritma KMP.

1. **Konstruksi Trie (Pohon Kata Kunci):** Semua kata kunci yang diberikan oleh pengguna dimasukkan ke dalam satu struktur data Trie. Setiap simpul (node) pada Trie merepresentasikan sebuah prefiks dari satu atau lebih kata kunci. Simpul akhir yang menandakan akhir dari sebuah kata kunci akan ditandai secara khusus sebagai "simpul keluaran" (output node).
2. **Pembuatan Link Kegagalan (Failure Links):** Ini adalah langkah kunci yang memberikan kecepatan pada Aho-Corasick. Untuk setiap simpul di Trie, sebuah "link kegagalan" dibuat. Link kegagalan dari sebuah simpul "u" (yang merepresentasikan string "s") akan menunjuk ke simpul lain di Trie yang merepresentasikan sufiks sejati terpanjang dari "s" yang juga merupakan prefiks dari kata kunci lain di dalam kamus. Link ini analog dengan tabel LPS pada KMP dan digunakan untuk menghindari penelusuran ulang pada teks saat terjadi ketidakcocokan.
3. **Penandaan Keluaran (Output):** Setiap simpul yang menandai akhir dari sebuah kata kunci akan dicatat. Jika beberapa kata kunci berbagi sufiks (misalnya, "react" dan "reactjs"), link keluaran tambahan dapat dibuat untuk memastikan semua kecocokan terdeteksi saat algoritma mencapai simpul tersebut.

Setelah automaton selesai dibangun, pencocokan pada teks CV dilakukan dalam satu kali lintasan (single pass) dari awal hingga akhir.

1. **Inisialisasi:** Proses dimulai dari simpul akar (root) pada automaton dan penunjuk pada karakter pertama dari teks CV.
2. **Iterasi Melalui Teks:** Algoritma membaca teks CV karakter per karakter dan bergerak melalui automaton sesuai dengan karakter yang dibaca.
 - **Transisi Berhasil:** Jika dari simpul saat ini terdapat transisi (anak) yang cocok dengan karakter pada teks, algoritma akan pindah ke simpul anak tersebut.

- **Transisi Gagal (Mismatch):** Jika tidak ada transisi yang cocok dari simpul saat ini, algoritma akan mengikuti **link kegagalan** dari simpul tersebut ke simpul baru, lalu mencoba kembali transisi dengan karakter teks yang sama. Proses ini diulang hingga transisi ditemukan atau kembali ke simpul akar. Penunjuk pada teks **tidak pernah mundur**, hanya status pada automaton yang berubah.

3. **Deteksi Kecocokan:** Setiap kali algoritma memasuki sebuah simpul baru (baik melalui transisi berhasil maupun link kegagalan), sistem akan memeriksa apakah simpul tersebut adalah "simpul keluaran".

- Jika ya, maka sebuah kecocokan dengan kata kunci yang bersangkutan telah ditemukan.
- Algoritma juga akan mengikuti rantai link keluaran (jika ada) dari simpul tersebut untuk mencatat kata kunci lain yang mungkin merupakan sufiks dari string yang baru saja cocok (contoh: menemukan "SQL" setelah menemukan "NoSQL"). Proses pencarian terus berlanjut hingga akhir teks untuk menemukan semua kemungkinan kecocokan.

3.3. Fitur Fungsional dan Arsitektur Aplikasi yang Dibangun

3.3.1. Fitur Fungsional

1. Fitur Pencarian

Fitur pencarian merupakan fitur utama dari aplikasi. Pengguna dapat masuk ke halaman pencarian dan mengisi informasi sesuai dengan kebutuhannya. Informasi tersebut antara lain, kata-kata kunci, jumlah hasil, dan algoritma yang diinginkan. Setelah mengirim informasi, pengguna diminta untuk menunggu sebentar selama sistem memproses seluruh data.

Saat data selesai diolah, pengguna diarahkan ke halaman hasil pencarian. Pengguna disuguhkan lama waktu pemrosesan dengan algoritma, dan calon-calon pekerja yang cocok dengan yang diinginkan pengguna. Untuk mempelajari hasil lebih lanjut, pengguna dapat menekan tombol “Summary” untuk melihat

rangkuman CV calon pekerja, dan tombol “Lihat CV” untuk melihat file CV asli calon pekerja.

2. Fitur Halaman Summary

Pengguna dapat menekan tombol “Summary” untuk mempelajari kemampuan dan pengalaman pekerja lebih lanjut. Halaman ini berisi rangkuman singkat calon pekerja, kemampuan, dan riwayat edukasi serta pengalamannya.

3. Fitur buka CV

Pengguna dapat menekan tombol “Lihat CV” untuk melihat dokumen CV asli calon pekerja. Sistem akan melacak dan membuka dokumen tersebut yang ada di data pengguna.

3.3.2. Arsitektur Aplikasi

Aplikasi ini dibangun menggunakan framework Flet, sebuah pustaka Python yang memungkinkan pengembangan antarmuka pengguna (UI) modern dan interaktif tanpa perlu menulis kode JavaScript atau Dart. Dengan pendekatan berbasis komponen, Flet memungkinkan developer membuat aplikasi web, desktop, maupun mobile menggunakan bahasa Python secara langsung. Dalam pembangunan aplikasi ini, digunakan struktur navigasi berbasis halaman seperti Beranda, Pencarian, Hasil Pencarian, Ringkasan CV, dan Tentang Kru. Perpindahan antar halaman dikelola melalui mekanisme `page.go` dan `page.views.append`, sementara data sementara seperti hasil pencarian dan waktu eksekusi disimpan di `page.session`, sehingga dapat dipertahankan meskipun pengguna berpindah halaman.

Antarmuka aplikasi dibentuk menggunakan komponen bawaan Flet seperti Container, Column, Row, dan Text, yang dikombinasikan dengan pengaturan style seperti padding, margin, color, dan ukuran font untuk menciptakan tampilan yang bersih, responsif, dan mudah digunakan. Di sisi fungsionalitas, aplikasi terhubung dengan backend Python yang menangani proses ekstraksi informasi dari CV, pencocokan keyword berdasarkan algoritma tertentu, dan pengembalian hasil ranking serta detail kecocokan.

Database aplikasi ini menggunakan MySQL, dengan dua tabel utama, yaitu ApplicantProfile dan ApplicationDetail. Tabel ApplicantProfile menyimpan informasi pribadi pelamar, sementara ApplicationDetail menyimpan informasi terkait lamaran pekerjaan, termasuk lokasi file CV. Pada tahap inisialisasi, aplikasi akan memastikan skema data sudah tersedia, dan jika belum, sistem akan membuat struktur tabel secara otomatis menggunakan perintah SQL. Sistem kemudian akan mengekstrak data teks dan mengisi data pelamar ke dalam database.

3.4. Contoh Ilustrasi Kasus.

Berikut adalah contoh ilustrasi kasus pemrosesan CV menggunakan algoritma KMP, BM, dan AC. Contoh ilustrasi kasus menggunakan salah satu CV yang terdapat dalam basis data dengan nama file **10030015.pdf** dan role ENGINEERING.

Teks diekstraksi dari file PDF dan dinormalisasi menjadi huruf kecil. Penulis akan menggunakan potongan teks berikut dari bagian "Professional Experience" sebagai Teks (T).

- Teks (T): "**operate electrical, mechanical, and hydraulic systems to test pumps, motors, and actuators for aircraft**"

Penulis akan mencari kata kunci electrical sebagai Pola (P) di dalam teks tersebut.

- Pola untuk BM dan KMP (P): "**electrical**"
- Pola untuk AC (P): "**electrical**", "**systems**", "**electric**", "**mechanical**"

3.4.1. KMP

Algoritma KMP menggunakan tabel LPS (*Longest Proper Prefix suffix*) atau *Border Function* untuk menghindari perbandingan yang tidak perlu saat terjadi ketidakcocokan.

Pra-pemrosesan - Membuat Tabel LPS

Tabel LPS untuk pola electrical adalah sebagai berikut. Karena tidak ada awalan yang juga merupakan akhiran dalam pola ini, semua nilai LPS adalah 0.

j	0	1	2	3	4	5	6	7	8	9
k	-1	0	1	2	3	4	5	6	7	8
b(k)	-	0	0	1	0	0	0	0	0	0

Proses Pencocokan

operate electrical, mechanical, and hydraulic systems ...

electrical

1

electrical

2

electrical

34

electrical

5

electrical

6

electrical

7

electrical

8

electrical

9

electrical

10

11 12 13 14 15 16 17 18 19

3.4.2. Boyer Moore

Algoritma Boyer-Moore membandingkan pola dari kanan ke kiri dan menggunakan Tabel *Last Occurrence* untuk melakukan lompatan besar.

electrical

Pra-pemrosesan - Membuat Tabel Lo

Char	e	l	c	t	r	i	a
Lo	2	9	7	4	5	6	8

Proses Pencocokan

operate electrical, mechanical, and hydraulic systems ...

electrical

2 1

electrical

3

electrical

13 12 11 10 9 8 7 6 5 4

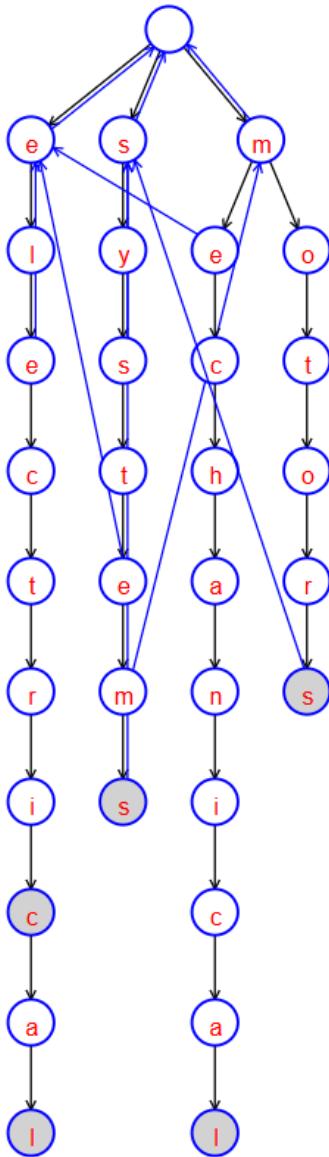
3.4.3. Aho-Corasick

Kekuatan utama Aho-Corasick adalah kemampuannya untuk mencari beberapa kata kunci sekaligus dalam satu kali proses. Oleh karena itu, kita akan menggunakan set kata kunci yang lebih kompleks.

Sistem akan membangun sebuah Trie dari kata kunci “electrical”, “systems”, “electric”, “mechanical”, “motors”. Setiap failure link akan menunjuk kembali ke root.

T = “**operate electrical, mechanical, and hydraulic systems to test pumps, motors, and actuators for aircraft**”

P: “electrical”, “systems”, “electric”, “mechanical”, “motors”



Gambar 3.4.3.1. Automaton Trie dari contoh kasus

(Sumber: <https://wiomoc.github.io/aho-corasick-viz/>)

Proses Pencocokan (Satu Kali Jalan)

Algoritma akan memindai teks dari kiri ke kanan, berpindah dari satu *state* ke *state* lain di dalam *Trie*.

1. **Teks: operate ...**
 - Algoritma memindai teks. Tidak ada awalan dari kata kunci yang cocok. *State* tetap berada di *root*.
2. **Teks: ...electrical...**
 - Saat pemindaian mencapai e di "electrical" (indeks 8), algoritma mulai bergerak menuruni *Trie*: e → l → e → c → t → r → i → c.
 - Setelah mencapai c terakhir (indeks 15), *state* berada di node akhir yang menandai kata kunci "electric". **Pencocokan ditemukan** untuk "electric" yang dimulai pada indeks 8.
 - Algoritma tidak berhenti. Ia melanjutkan ke karakter berikutnya, a lalu 1.
 - Setelah mencapai l terakhir (indeks 17), *state* berada di node akhir untuk kata kunci "electrical". **Pencocokan kedua ditemukan** untuk "electrical" yang juga dimulai pada indeks 8.
3. **Teks: ..., mechanical, ...**
 - Pencarian dilanjutkan. Saat mencapai m di "mechanical" (indeks 20), ia mulai menuruni cabang lain dari *Trie*.
 - Setelah l terakhir (indeks 29), ia mencapai node akhir untuk "mechanical". **Pencocokan ditemukan** untuk "mechanical" pada indeks 20.
4. **Teks: ...and hydraulic systems...**
 - Pencarian dilanjutkan. Saat mencapai s di "systems" (indeks 45), ia menuruni cabang ketiga.
 - Setelah s terakhir (indeks 51), ia mencapai node akhir untuk "systems". **Pencocokan ditemukan** untuk "systems" pada indeks 45.
5. **Teks: ...to test pumps, motors...**
 - Pencarian dilanjutkan. Saat mencapai m di "motors" (indeks 59), ia menuruni cabang yang sama dengan "mechanical" pada awalnya, namun kemudian beralih ke cabang m → o.
 - Setelah s terakhir (indeks 64), ia mencapai node akhir untuk "motors". **Pencocokan ditemukan** untuk "motors" pada indeks 59.

Dengan hanya **satu kali pemindaian teks**, Aho-Corasick berhasil menemukan semua kemunculan dari semua kata kunci, termasuk kata kunci yang tumpang tindih seperti "electric" dan "electrical". Ini menunjukkan efisiensi superiornya dibandingkan menjalankan KMP atau BM berulang kali untuk setiap kata kunci.

BAB IV – IMPLEMENTASI DAN PENGUJIAN

4.1. Spesifikasi Teknis Program

Struktur Data

Sistem menggunakan basis data MySQL untuk menyimpan informasi persisten mengenai pelamar dan aplikasi mereka. Skema basis data terdiri dari dua tabel utama:

- ApplicantProfile: Tabel ini menyimpan data pribadi yang sensitif dari setiap pelamar. Untuk menjaga keamanan, seluruh data pribadi dienkripsi sebelum disimpan. Kolom-kolomnya meliputi:
 - applicant_id (INT, Primary Key): ID unik untuk setiap pelamar.
 - first_name, last_name, date_of_birth, address, phone_number, email (TEXT): Kolom-kolom ini menyimpan data pribadi yang telah dienkripsi. Tipe data TEXT digunakan untuk memastikan string hasil enkripsi yang lebih panjang dapat ditampung.
 - identifier (VARCHAR, UNIQUE): ID unik yang berasal dari nama file CV, digunakan untuk mencegah duplikasi data saat proses seeding.
- ApplicationDetail: Tabel ini menyimpan detail spesifik untuk setiap lamaran yang diajukan, menghubungkan seorang pelamar dengan sebuah file CV. Kolom-kolomnya meliputi:
 - detail_id (INT, Primary Key): ID unik untuk setiap detail aplikasi.
 - applicant_id (INT, Foreign Key): Menghubungkan ke tabel ApplicantProfile.
 - application_role (VARCHAR): Menyimpan jabatan yang dilamar (misalnya, "Software Engineer"), sesuai dengan data yang disediakan.
 - cv_path (TEXT): Menyimpan path relatif ke file CV di dalam direktori data/.

Selama eksekusi, program menggunakan beberapa struktur data Python untuk mengelola informasi secara efisien:

- Kamus (Dictionary) untuk Hasil Pencarian: Ini adalah struktur data utama yang digunakan untuk komunikasi antara backend (search_engine.py) dan frontend (GUI).

- Kamus (Dictionary) untuk Detail CV: Saat pengguna meminta ringkasan detail, fungsi `get_cv_summary_details` mengembalikan sebuah kamus yang berisi hasil ekstraksi Regex, dengan struktur:

Fungsi dan Prosedur

- `src/algorithms/`

[KMP.py](#)

```
● ○ ●

1  class KMP:
2      def __init__(self, pattern: str):
3          self.pattern = pattern
4          self.prefix_table = self._compute_prefix_table()
5
6      def _compute_prefix_table(self) -> list[int]:
7          m = len(self.pattern)
8          prefix_table = [0] * m
9          j = 0 # length of previous longest prefix suffix
10
11     for i in range(1, m):
12         while j > 0 and self.pattern[i] != self.pattern[j]:
13             j = prefix_table[j - 1]
14
15         if self.pattern[i] == self.pattern[j]:
16             j += 1
17             prefix_table[i] = j
18         else:
19             prefix_table[i] = 0
20
21     return prefix_table
22
23     def search(self, text: str) -> list[int]:
24         n = len(text)
25         m = len(self.pattern)
26         j = 0 # index for pattern
27         occurrences = []
28
29         for i in range(n):
30             while j > 0 and text[i] != self.pattern[j]:
31                 j = self.prefix_table[j - 1]
32
33             if text[i] == self.pattern[j]:
34                 j += 1
35
36             if j == m:
37                 occurrences.append(i - m + 1)
38                 j = self.prefix_table[j - 1]
39
40     return occurrences
```

[BM.py](#)

```
● ○ ● ●  
1 class BoyerMoore:  
2     def __init__(self, pattern: str):  
3         self.pattern = pattern  
4         self.m = len(pattern)  
5         self.last_occurrence_table = self._build_last_occurrence_table()  
6  
7     def _build_last_occurrence_table(self) -> dict[str, int]:  
8         table = {}  
9         for i, char in enumerate(self.pattern):  
10             table[char] = i  
11         return table  
12  
13     def search(self, text: str) -> list[int]:  
14         pattern = self.pattern  
15         m = self.m  
16         last_occurrence_table = self.last_occurrence_table  
17  
18         n = len(text)  
19         occurrences = []  
20  
21         if not text or not pattern or m > n:  
22             return []  
23  
24         s = 0  
25         while s <= n - m:  
26             j = m - 1  
27  
28             while j >= 0 and pattern[j] == text[s + j]:  
29                 j -= 1  
30  
31             if j < 0:  
32                 occurrences.append(s)  
33                 s += m - last_occurrence_table.get(text[s + m], -1) if s + m < n else 1  
34  
35             else:  
36                 bad_char = text[s + j]  
37                 shift = j - last_occurrence_table.get(bad_char, -1)  
38                 s += max(1, shift)  
39  
40         return occurrences
```

[Levenshtein.py](#)

```
● ● ●
```

```
1 # --- Levenshtein Distance and Similarity Utilities ---
2 def levenshtein_distance(s1: str, s2: str) -> int:
3     """Calculates the Levenshtein distance between two strings."""
4     m, n = len(s1), len(s2)
5     dp = [[0] * (n + 1) for _ in range(m + 1)]
6     for j in range(n + 1): dp[0][j] = j
7     for i in range(m + 1): dp[i][0] = i
8     for i in range(1, m + 1):
9         for j in range(1, n + 1):
10            cost = 0 if s1[i - 1] == s2[j - 1] else 1
11            dp[i][j] = min(dp[i - 1][j] + 1,           # Deletion
12                           dp[i][j - 1] + 1,           # Insertion
13                           dp[i - 1][j - 1] + cost) # Substitution
14    return dp[m][n]
15
16 def calculate_levenshtein_similarity(s1: str, s2: str) -> float:
17     # Calculates a similarity score (0.0 to 1.0) based on Levenshtein distance.
18     if not s1 and not s2: return 1.0
19     if not s1 or not s2: return 0.0
20     max_len = max(len(s1), len(s2))
21     if max_len == 0: return 1.0
22     distance = levenshtein_distance(s1, s2)
23     return 1.0 - (distance / max_len)
```

[Aho-Corasick.py](#)

```

1 import collections
2
3 class TrieNode:
4     def __init__(self):
5         self.children = collections.defaultdict(TrieNode)
6         self.failure_link = None
7         self.output = []
8
9 class AhoCorasick:
10    def __init__(self, keywords = list[str]):
11        self.root = TrieNode()
12        self.keywords = keywords
13        self._build_trie()
14        self._build_failure_links()
15
16    def _build_trie(self):
17        for keyword in self.keywords:
18            node = self.root
19            for char in keyword:
20                node = node.children[char]
21                node.output.append(keyword)
22
23    def _build_failure_links(self):
24        queue = collections.deque()
25
26        for char, node in self.root.children.items():
27            node.failure_link = self.root
28            queue.append(node)
29
30        while queue:
31            current_node = queue.popleft()
32
33            for char, next_node in current_node.children.items():
34                fail_node = current_node.failure_link
35
36                while fail_node is not None and char not in fail_node.children:
37                    fail_node = fail_node.failure_link
38
39                if fail_node:
40                    next_node.failure_link = fail_node.children[char]
41                else:
42                    next_node.failure_link = self.root
43
44                if next_node.failure_link:
45                    next_node.output.extend(next_node.failure_link.output)
46
47            queue.append(next_node)
48
49    def search(self, text: str) -> dict[str, list[int]]:
50        results = collections.defaultdict(list)
51        current_node = self.root
52
53        for i, char in enumerate(text):
54            while current_node is not None and char not in current_node.children:
55                current_node = current_node.failure_link
56
57            if current_node is None:
58                current_node = self.root
59                continue
60
61            current_node = current_node.children[char]
62
63            if current_node.output:
64                for keyword in current_node.output:
65                    start_index = i - len(keyword) + 1
66                    results[keyword].append(start_index)
67
68        return dict(results)

```

- src/core/

pdf_to_text.py

```
1  import fitz # PyMuPDF
2  import os
3  import re
4  import unicodedata # For advanced text cleaning
5
6  def _clean_and_normalize_text(text: str) -> str:
7      if not text:
8          return ""
9      # Normalize Unicode characters to their simplest form (NFKC).
10     text = unicodedata.normalize('NFKC', text)
11     # Replace common non-standard bullets or symbols with a standard one or remove them.
12     text = re.sub(r'[•••]', r'*', text)
13     # Remove any remaining non-ASCII characters. This is an aggressive but effective
14     text = text.encode('ascii', 'ignore').decode('ascii')
15     # Standardize whitespace. Replaces multiple spaces/newlines/tabs with a single space.
16     text = re.sub(r'\s+', ' ', text)
17     return text.strip()
18
19
20 def extract_text_from_pdf(pdf_path: str) -> tuple[str | None, str | None]:
21     if not os.path.exists(pdf_path):
22         print(f"[pdf_to_text] Error: File not found at '{pdf_path}'")
23         return None, None
24
25     try:
26         doc = fitz.open(pdf_path)
27
28         raw_page_texts = []
29         for page in doc:
30             raw_page_texts.append(page.get_text("text"))
31         doc.close()
32
33         if not raw_page_texts:
34             return "", ""
35
36         # Join pages and perform basic normalization. This preserves paragraph structure.
37         full_text_raw = "\n\n".join(raw_page_texts)
38         text_for_regex = unicodedata.normalize('NFKC', full_text_raw).strip()
39
40         text_for_pattern_matching = _clean_and_normalize_text(full_text_raw)
41
42         return text_for_regex, text_for_pattern_matching
43
44     except Exception as e:
45         print(f"[pdf_to_text] Error processing PDF file '{pdf_path}': {e}")
46         return None, None
47
48
```

[regex.py](#)

```
1 import re
2
3
4 HEADERS = [
5     "experience", "work experience", "professional experience", "relevant experience",
6     "education", "education and training",
7     "skills", "technical skills", "skill highlights",
8     "summary", "professional summary", "summary of skills", "technical summary",
9     "highlights", "accomplishments",
10    "certifications", "certifications and credentials",
11    "languages", "language skills",
12    "profile", "professional profile", "activities and honors"
13 ]
14
15 def extract_all_cv_details(text: str) -> dict:
16     sections = extract_sections(text)
17     details = reformat_sections(sections)
18     return details
19
20 def clean_text(text: str) -> str:
21     text = re.sub(r">>+", "", text)
22     text = re.sub(r"\n\n";, " ", text)
23     text = re.sub(r"\r", "", text)
24     text = re.sub(r"+", " ", text)
25     text = re.sub(r"\n+", "\n", text)
26     return text.strip()
27
28 def extract_sections(text: str, target_sections=None):
29     if target_sections is None:
30         target_sections = ["summary", "skills", "education", "experience"]
31
32     headers_pattern = "|".join([re.escape(h) for h in HEADERS])
33     pattern = rf"(?im)^{s*({headers_pattern})}\s*$"
34
35     matches = list(re.finditer(pattern, text, flags=re.MULTILINE))
36     sections = {}
37
38     for i, match in enumerate(matches):
39         header = match.group(1).lower()
40         start = match.end()
41         end = matches[i+1].start() if i+1 < len(matches) else len(text)
42         # Normalisasi header ke target section
43         for section in target_sections:
44             if section in header:
45                 section_key = section
46                 break
47             else:
48                 continue
49         section_text = text[start:end].strip()
50         sections[section_key] = clean_text(section_text)
51
52     for section in target_sections:
53         if section not in sections:
54             sections[section] = ""
55
56     return sections
57
58
59 def reformat(text: str) -> str:
60     text = re.sub(r'^\x00\x7f]+', '', text)
61     text = text.replace('\n', ' ')
62     text = re.sub(r',+', ',', text)
63     return text.strip(',')
64
65
66 def reformat_sections(sections: dict) -> dict:
67     return {
68         "summary": reformat(sections.get("summary", "")),
69         "skills": reformat(sections.get("skills", "")),
70         "education": reformat(sections.get("education", "")),
71         "experience": reformat(sections.get("experience", ""))
72     }
73
74 def validate_email(text: str):
75     email_match = re.search(r"\b[a-z0-9]+(?:[-\.\.][a-z0-9]+)@[a-z0-9]+(?:-[a-z0-9]+)\.[a-z]{2,}\b", text)
76     return email_match.group(0) if email_match else None
```

search_engine.py

```

1   # KMP / BOYER-MOORE LOGIC
2   else:
3       SearcherClass = KMP if algorithm_choice.upper() == "KMP" else BoyerMoore
4       for cv_info in cv_data_for_processing:
5           applicant_id = cv_info['applicant_id']
6           cv_pm_text = cv_texts.get(applicant_id, {}).get('pm', '')
7           if not cv_pm_text: continue
8
9           matched_details = defaultdict(int)
10          for keyword in input_keywords:
11              searcher = SearcherClass(keyword)
12              occurrences = searcher.search(cv_pm_text)
13              if occurrences:
14                  matched_details[keyword] = len(occurrences)
15
16          if matched_details:
17              results_in_progress[applicant_id] = {
18                  'applicant_id': applicant_id, 'detail_id': cv_info['detail_id'],
19                  'name': cv_info['name'], 'cv_path': cv_info['cv_path'],
20                  'matched_keywords_details': matched_details
21              }
22
23      exact_match_duration = time.perf_counter() - exact_timer_start
24
25      # FUZZY MATCHING
26      fuzzy_timer_start = time.perf_counter()
27      fuzzy_match_performed = False
28
29      for cv_info in cv_data_for_processing:
30          applicant_id = cv_info['applicant_id']
31          current_result = results_in_progress.get(applicant_id)
32
33          exact_matches = set(current_result['matched_keywords_details'].keys()) if current_result else set()
34          unmatched_keywords = set(input_keywords) - exact_matches
35          if not unmatched_keywords: continue
36
37          cv_words = cv_texts.get(applicant_id, {}).get('pm', '').split()
38          if not cv_words: continue
39
40          for keyword in unmatched_keywords:
41              best_similarity = 0.0
42              for word in cv_words:
43                  similarity = calculate_levenshtein_similarity(keyword, word)
44                  if similarity >= levenshtein_similarity_threshold and similarity > best_similarity:
45                      best_similarity = similarity
46
47          if best_similarity > 0:
48              fuzzy_match_performed = True
49              fuzzy_key = f'[keyword] ({fuzzy ~int(best_similarity*100)})'
50
51          if applicant_id not in results_in_progress:
52              results_in_progress[applicant_id] = {
53                  'applicant_id': applicant_id, 'detail_id': cv_info['detail_id'],
54                  'name': cv_info['name'], 'cv_path': cv_info['cv_path'],
55                  'matched_keywords_details': defaultdict(float)
56              }
57
58          # Store the similarity score as the value for the fuzzy match
59          results_in_progress[applicant_id]['matched_keywords_details'][fuzzy_key] = best_similarity
60
61      fuzzy_match_duration = (time.perf_counter() - fuzzy_timer_start) if fuzzy_match_performed else 0.0
62
63      # FINAL SCORING & RANKING
64      final_results = list(results_in_progress.values())
65
66      for result in final_results:
67          details = result['matched_keywords_details']
68
69          exact_matches = {k: v for k, v in details.items() if not k.endswith('')}
70          fuzzy_matches = {k: v for k, v in details.items() if k.endswith('')}
71
72          # Main score from unique keywords (1 point per exact, similarity score per fuzzy)
73          score = (len(exact_matches) * 1.0) + sum(fuzzy_matches.values())
74
75          # Add a small tie-breaker bonus based on the frequency of exact matches
76          frequency_bonus = sum(exact_matches.values()) * FREQUENCY_BONUS_WEIGHT
77          result['score'] = score + frequency_bonus
78
79          result['matched_keywords_count'] = len(details)
80          result['matched_keywords_details'] = dict(details)
81
82      final_results.sort(key=lambda x: x.get('score', 0), reverse=True)
83      top_results = final_results[:top_n]
84
85      total_processing_time = time.perf_counter() - total_timer_start
86      timing_info = {
87          'exact_match_time': round(exact_match_duration, 4),
88          'fuzzy_match_time': round(fuzzy_match_duration, 4),
89          'total_processing_time': round(total_processing_time, 4),
90          'cvs_processed': len(cv_data_for_processing),
91          'status_message': f'Search complete. Found {len(final_results)} potential matches.'
92      }
93
94      return top_results, timing_info
95
96  def get_cv_summary_details(cv_path: str) -> dict | None:
97      # Extracts detailed structured information for the summary page.
98      full_path = get_cv_path(cv_path)
99      if not full_path:
100         return None
101
102     raw_text, _ = extract_text_from_pdf(full_path)
103     if raw_text is None:
104         return None
105
106     return extract_all_cv_details(raw_text)

```

- src/model/

database.py

```

1 # MySQL connection logic
2
3 import mysql.connector
4 from contextlib import contextmanager
5 import os
6 from dotenv import load_dotenv
7
8 # load environment variables
9 load_dotenv()
10
11 # CONNECTION CONFIGURATION
12 DB_CONFIG = {
13     'host': os.getenv('ATS_DB_HOST', 'localhost'),
14     'user': os.getenv('ATS_DB_USER', 'root'),
15     'password': os.getenv('ATS_DB_PASS', ''),
16     'database': os.getenv('ATS_DB_NAME', 'ats_db'),
17     'charset': 'utf8mb4',
18     'use_unicode': True
19 }
20
21 # CONNECTION FUNCTION
22 def get_db_connection():
23     try:
24         conn = mysql.connector.Connect(**DB_CONFIG)
25         return conn
26     except Exception as e:
27         print(f"[Database] Exception connecting to MySQL: {e}")
28         return None
29
30 # CONTEXT MANAGER FOR CURSOR
31 @contextmanager
32 def get_db_cursor(commit: bool = True):
33     conn = get_db_connection()
34     if conn is None:
35         yield None
36         return
37
38     cursor = conn.cursor(buffered=True)
39     try:
40         yield cursor
41         if commit:
42             conn.commit()
43     except Exception as e:
44         conn.rollback()
45         print(f"[database.py][get_db_cursor] Exception when executing query: {e}")
46         raise
47     finally:
48         cursor.close()
49         conn.close()
50
51 # HELPER FUNCTIONS
52 def execute_query(query: str, params: tuple = None):
53     with get_db_cursor(commit=True) as cursor:
54         if cursor is None:
55             raise Exception("Cannot connect to the database.")
56         if params:
57             cursor.execute(query, params or ())
58         else:
59             cursor.execute(query)
60
61 def fetch_all(query: str, params: tuple = None):
62     with get_db_cursor(commit=False) as cursor:
63         if cursor is None:
64             raise Exception("Cannot connect to the database.")
65         if params:
66             cursor.execute(query, params)
67         else:
68             cursor.execute(query)
69         rows = cursor.fetchall()
70     return rows
71
72 def fetch_one(query: str, params: tuple = None) -> tuple | None:
73     with get_db_cursor(commit=False) as cursor:
74         if cursor is None:
75             raise ConnectionError("Cannot connect to the database.")
76         if params:
77             cursor.execute(query, params)
78         else:
79             cursor.execute(query)
80         row = cursor.fetchone()
81     return row
82
83 # UTILITY FUNCTION TO INSERT AND GET ID
84 def insert_and_get_id(query: str, params: tuple = None) -> int:
85     with get_db_cursor(commit=True) as cursor:
86         if cursor is None:
87             raise ConnectionError("Cannot connect to the database to insert data.")
88
89         cursor.execute(query, params or ())
90         # cursor.lastrowid is available immediately after execute for an INSERT
91         return cursor.lastrowid

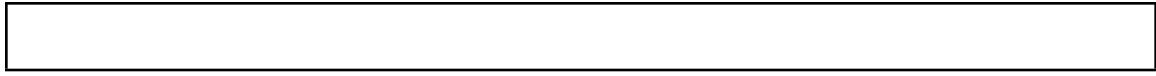
```

models.py

```

1  from .database import execute_query, fetch_all, fetch_one, insert_and_get_id
2  from .datetime import date
3
4  from ..utils.encryption import encrypt, decrypt
5
6  # ----- CREATE TABLE IF NOT EXISTS -----
7  def create_tables():
8      # Base64 encoding increases string length, so we change sensitive fields to TEXT
9      # to ensure the encrypted strings fit without being truncated.
10
11     sql_profile = """
12         CREATE TABLE IF NOT EXISTS ApplicantProfile (
13             applicant_id INT AUTO_INCREMENT PRIMARY KEY,
14             first_name TEXT DEFAULT NULL,
15             last_name TEXT DEFAULT NULL,
16             address TEXT DEFAULT NULL,
17             phone_number TEXT DEFAULT NULL,
18             email TEXT DEFAULT NULL,
19             created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
20             identifier VARCHAR(255) UNIQUE
21         ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
22     """
23
24     execute_query(sql_profile)
25
26     sql_application = """
27         CREATE TABLE IF NOT EXISTS ApplicationDetail (
28             detail_id INT AUTO_INCREMENT PRIMARY KEY,
29             applicant_id INT NOT NULL,
30             application_role VARCHAR(100) DEFAULT NULL,
31             cv_path TEXT DEFAULT NULL,
32             applied_date DATE DEFAULT NULL,
33             FOREIGN KEY (applicant_id) REFERENCES ApplicantProfile(applicant_id) ON DELETE CASCADE
34         ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
35     """
36
37     execute_query(sql_application)
38     print("[models.py] application create_tables execution attempted.")
39
40     # ----- INSERT FUNCTIONS -----
41
42     def insert_applicant_profile(
43         first_name: str = None, last_name: str = None, date_of_birth: date = None,
44         address: str = None, phone_number: str = None, email: str = None,
45         identifier: str = None
46     ) -> int | None:
47
48         # Encrypt all sensitive Personally Identifiable Information (PII) before inserting.
49         encrypted_first_name = encrypt(first_name)
50         encrypted_last_name = encrypt(last_name)
51         encrypted_dob = encrypt(str(date_of_birth)) if date_of_birth else None
52         encrypted_address = encrypt(address)
53         encrypted_phone = encrypt(phone_number)
54         encrypted_email = encrypt(email)
55
56         sql = """
57             INSERT INTO ApplicantProfile (first_name, last_name, date_of_birth, address, phone_number, email, identifier)
58             VALUES (%s, %s, %s, %s, %s, %s);
59         """
60
61         params = (
62             encrypted_first_name, encrypted_last_name, encrypted_dob,
63             encrypted_address, encrypted_phone, encrypted_email, identifier
64         )
65
66         try:
67             new_id = insert_and_get_id(sql, params)
68             return new_id
69         except Exception as e:
70             print(f"[models.py][insert_applicant_profile] Failed: {e}")
71         raise
72
73     def insert_application_detail(
74         applicant_id: int, cv_path: str, application_role: str = None,
75         applied_date: date = None
76     ) -> int | None:
77         if applied_date is None:
78             applied_date = date.today()
79
80         sql = """
81             INSERT INTO ApplicationDetail (applicant_id, cv_path, application_role, applied_date)
82             VALUES (%s, %s, %s, %s);
83         """
84
85         params = (applicant_id, cv_path, application_role, applied_date)
86         return insert_and_get_id(sql, params)
87
88     # FETCH FUNCTIONS
89     def fetch_all_cv_details():
90         # This function does not need changes.
91         sql = "SELECT detail_id, applicant_id, application_role, cv_path, applied_date FROM ApplicationDetail;"
92         return fetch_all(sql)
93
94     def fetch_applicant_by_id(applicant_id_val: int) -> dict | None:
95         sql = """
96             SELECT applicant_id, first_name, last_name, date_of_birth, address, phone_number, email, created_at, identifier
97             FROM ApplicantProfile
98             WHERE applicant_id = %s;
99         """
100
101         params = (applicant_id_val,)
102         row = fetch_one(sql, params)
103
104         if row:
105             # Decrypt all sensitive fields after fetching from the database.
106             decrypted_dob = None
107             decrypted_dob_str = decrypt(row[4])
108             if decrypted_dob_str and decrypted_dob_str != 'None':
109                 try:
110                     # Convert the decrypted date string back into a date object.
111                     decrypted_dob = date.fromisoformat(decrypted_dob_str)
112                 except (ValueError, TypeError):
113                     decrypted_dob = None # Handle cases where date might be malformed
114
115             return {
116                 "applicant_id": row[0],
117                 "first_name": decrypt(row[1]),
118                 "last_name": decrypt(row[2]),
119                 "date_of_birth": decrypted_dob,
120                 "address": decrypt(row[4]),
121                 "phone_number": decrypt(row[5]),
122                 "email": decrypt(row[6]),
123                 "created_at": row[7],
124                 "identifier": row[8]
125             }
126
127         return None

```



- src/utils/

data_seeding.py

```

1 import os
2 from datetime import date
3 from faker import Faker
4
5 from ..model.models import (
6     insert_application_profile,
7     insert_application_detail,
8 )
9
10 from ..model.database import fetch_one
11
12 from ..utils.file_handler import get_all_pdf_filenames, get_cv_path
13
14 fake = Faker()
15
16 THIS_DIR = os.path.dirname(os.path.abspath(__file__))
17 DATA_DIR = os.path.abspath(os.path.join(THIS_DIR, '..', '..', 'data'))
18
19 def _split_name(full_name: str) -> tuple[None | None, str | None]:
20     # Helper to split a full name into first and last names.
21     parts = full_name.split()
22
23     if not parts:
24         return None, None
25     first_name = parts[0]
26     last_name = "+".join(parts[1:]) if len(parts) > 1 else None
27     return first_name, last_name
28
29 def seed_with_dummy(data):
30     print("[SeedDummy] Looking for PDF files...")
31
32     pdf_files = get_all_pdf_filenames()
33
34     if not pdf_files:
35         print("[SeedDummy] No PDF files found to process.")
36         return
37
38     total_files_processed = 0
39     new_applications_created = 0
40
41     for filename in pdf_files:
42         total_files_processed += 1
43         identifier = filename[-4]
44         full_pdf_path = get_cv_path(filename)
45         if not full_pdf_path:
46             print("\n[SeedDummy] Skipping file {} because path could not be resolved.".format(filename))
47             continue
48
49         print("\n[SeedDummy] Processing file ({}/{len(pdf_files)}): {}".format(total_files_processed, filename))
50
51         # --- Step 1: Get or Create Applicant Profile ---
52         applicant_id = None
53         try:
54             existing_profile = fetch_one(
55                 "SELECT applicant_id FROM ApplicantProfile WHERE identifier = %s",
56                 (identifier,))
57
58             if existing_profile:
59                 applicant_id = existing_profile[0]
60                 print("[Info] Existing ApplicantProfile found with id={applicant_id}.")
61             else:
62                 # Generate and insert a new applicant profile
63                 first_name, last_name = _split_name(fake.name()) # Assuming _split_name helper exists
64                 applicant_id = insert_application_profile(
65                     first_name=first_name,
66                     last_name=last_name,
67                     date_of_birth=fake.date_of_birth(minimum_age=22, maximum_age=60),
68                     address=fake.address().replace('\n', ' '),
69                     phone_number=fake.phone_number(),
70                     email=fake.email(),
71                     identifier=identifier)
72                 print("[Insert] New ApplicantProfile created with id={applicant_id}.")
73
74         except Exception as e:
75             print("[Error] Failed during ApplicantProfile processing: ({})".format(e))
76             continue # Skid to the next file
77
78         # --- Step 2: Get or Create Application Detail (The Corrected Logic) ---
79         if applicant_id is None:
80             print("[Warning] Skipping ApplicationDetail for '{}' as applicant_id was not determined.".format(filename))
81             continue
82
83         try:
84             # check if application already exists
85             application_exists = fetch_one(
86                 "SELECT detail_id FROM ApplicationDetail WHERE applicant_id = %s AND cv_path = %s",
87                 (applicant_id, full_pdf_path))
88
89             if application_exists:
90                 print("[Info] ApplicationDetail for this CV already exists (detail_id={application_exists[0]}). Skipping.")
91
92             else:
93                 application.role_val = fake.job() if fake.boolean(chance_of_getting_true=75) else None
94
95                 print("[Info] Creating new ApplicationDetail for applicant_id={applicant_id}")
96                 detail_id = insert_application_detail(
97                     applicant_id=applicant_id,
98                     cv_path=full_pdf_path,
99                     application_role=application.role_val,
100                     applied_date=date.today())
101
102                 if detail_id:
103                     print("[Insert] New ApplicationDetail created: detail_id={detail_id}")
104                     new_applications_created += 1
105                 else:
106                     print("[Error] Failed to insert ApplicationDetail, received None ID.")
107
108         except Exception as e:
109             print("[Error] Failed to insert ApplicationDetail for applicant_id ({})".format(applicant_id))
110             continue
111
112         print("\n[SeedDummy] Finished processing. Processed {} PDF files. Created {} new application entries.".format(total_files_processed, new_applications_created))
113
114 if __name__ == '__main__':
115     print("Starting dummy data seeding process...")
116     seed_with_dummy(data)
117     print("Dummy data seeding process completed.")

```

encryption.py

```
 1 import os
 2 import base64
 3 from dotenv import load_dotenv
 4
 5 # Load environment variables to get the secret key
 6 load_dotenv()
 7
 8 # Fetch the secret key from environment variables.
 9 SECRET_KEY = os.getenv('ATS_ENCRYPTION_KEY', 'default_super_secret_key_that_is_long')
10
11 def _vigenere(text: str, key: str, encrypt: bool = True) -> str:
12     """Internal function to perform Vigenere encryption or decryption."""
13     processed_text = ""
14     key_index = 0
15     key_len = len(key)
16     for char in text:
17         char_code = ord(char)
18         key_char_code = ord(key[key_index % key_len])
19         if encrypt:
20             encrypted_code = (char_code + key_char_code) % 256
21             processed_text += chr(encrypted_code)
22         else:
23             decrypted_code = (char_code - key_char_code + 256) % 256
24             processed_text += chr(decrypted_code)
25         key_index += 1
26     return processed_text
27
28 def encrypt(plaintext: any) -> str | None:
29     """
30     Encrypts a plaintext string. If the input is not a string, it returns it
31     as-is without causing an error.
32     """
33     # --- FIX: More robust type checking ---
34     # Only attempt to encrypt if the input is a non-empty string.
35     if not isinstance(plaintext, str) or not plaintext:
36         # This will handle None, empty strings, lists, numbers, etc. gracefully.
37         return plaintext
38
39     try:
40         encrypted_text = _vigenere(plaintext, SECRET_KEY, encrypt=True)
41         encrypted_bytes = encrypted_text.encode('utf-8', 'surrogatepass')
42         base64_bytes = base64.b64encode(encrypted_bytes)
43         base64_string = base64_bytes.decode('utf-8')
44         return base64_string
45     except Exception as e:
46         # Added a debug print to see what data caused the failure.
47         print(f"[Cipher] Encryption failed for value '{plaintext}' of type {type(plaintext)}: {e}")
48         return plaintext # Return original text on failure
49
50 def decrypt(ciphertext: any) -> str | None:
51     """
52     Decrypts a ciphertext string. If the input is not a string, it returns it
53     as-is without causing an error.
54     """
55     # --- FIX: More robust type checking ---
56     if not isinstance(ciphertext, str) or not ciphertext:
57         return ciphertext
58
59     try:
60         base64_bytes = ciphertext.encode('utf-8')
61         encrypted_bytes = base64.b64decode(base64_bytes)
62         encrypted_text = encrypted_bytes.decode('utf-8', 'surrogatepass')
63         decrypted_text = _vigenere(encrypted_text, SECRET_KEY, encrypt=False)
64         return decrypted_text
65     except Exception:
66         # If decryption fails (e.g., it was never encrypted), return the original text.
67         # This prevents crashes if you have mixed encrypted/unencrypted data.
68         return ciphertext
69
```

file_handler.py

```
 1 import os
 2 import subprocess
 3 import sys
 4
 5 PROJECT_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), '.', '..'))
 6 DATA_DIR = os.path.join(PROJECT_ROOT, 'data')
 7
 8 def get_cv_path(relative_path: str) -> str | None:
 9     if not os.path.exists(DATA_DIR):
10         print(f"Data directory does not exist: {DATA_DIR}", file=sys.stderr)
11         return None
12     full_path = os.path.join(DATA_DIR, os.path.basename(relative_path))
13
14     if os.path.exists(full_path):
15         return full_path
16     else:
17         print(f"Warning: CV file not found at {full_path}")
18         return None
19
20 def get_all_pdf_filenames() -> list[str]:
21     if not os.path.isdir(DATA_DIR):
22         print(f"Error: Data directory not found at {DATA_DIR}", file=sys.stderr)
23         return []
24
25     # List comprehension to find all files ending with .pdf (case-insensitive)
26     return [f for f in os.listdir(DATA_DIR) if f.lower().endswith('.pdf')]
27
28
29 def open_file_with_default_app(filepath: str):
30     # Opens a file with system's default application (for "View CV" feature)
31     if not filepath or not os.path.exists(filepath):
32         print(f"Error: Cannot open file. Path doesn't exist: {filepath}")
33         return
34
35     try:
36         if sys.platform.startswith('win32'):
37             os.startfile(filepath)
38         elif sys.platform.startswith('darwin'): # macOS
39             subprocess.run(['open', filepath], check=True)
40         else: # linux
41             subprocess.run(['xdg-open', filepath], check=True)
42     except Exception as e:
43         print(f"Error opening file {filepath}: {e}", file=sys.stderr)
44
```

test_threshold.py

```

● ● ●
1  from algorithms.levenshtein import calculate_levenshtein_similarity
2
3  TEST_CASES = {
4      "python": [("python", True), ("pyhton", True), ("pyton", True), ("pyhton", True), ("java", False)],
5      "experience": [("experience", True), ("experence", True), ("exprience", True), ("expert", False)],
6      "software": [("software", True), ("software", True), ("softwares", True), ("hardware", False)],
7  }
8
9  # Thresholds to evaluate
10 THRESHOLDS_TO_TEST = [0.70, 0.75, 0.80, 0.85, 0.90, 0.95]
11
12 def run_test():
13     best_f1 = 0
14     best_threshold = None
15
16     for threshold in THRESHOLDS_TO_TEST:
17         # Initialize counters for true positives, false positives, false negatives, true negatives
18         tp = fp = fn = tn = 0
19
20         for correct, variations in TEST_CASES.items():
21             for variation, should_match in variations:
22                 # Calculate similarity (assumed to be a float between 0 and 1, where 1 is identical)
23                 sim = calculate_levenshtein_similarity(correct, variation)
24                 is_match = sim >= threshold
25
26                 # Update counters based on match outcome and ground truth
27                 if is_match and should_match:
28                     tp += 1
29                 elif is_match and not should_match:
30                     fp += 1
31                 elif not is_match and should_match:
32                     fn += 1
33                 else:
34                     tn += 1
35
36                 # Calculate metrics with safeguards for zero denominators
37                 precision = tp / (tp + fp) if tp + fp > 0 else 1 # No positive predictions -> precision undefined, set to 1
38                 recall = tp / (tp + fn) if tp + fn > 0 else 0 # No true positives -> recall 0
39                 f1 = 2 * precision * recall / (precision + recall) if precision + recall > 0 else 0
40
41                 # Display results for current threshold
42                 print(f"Threshold {threshold:.2f}: TP={tp}, FP={fp}, FN={fn}, TN={tn}, "
43                       f"Precision={precision:.2f}, Recall={recall:.2f}, F1={f1:.2f}")
44
45                 # Track the best threshold based on F1-score
46                 if f1 > best_f1:
47                     best_f1 = f1
48                     best_threshold = threshold
49
50                 # Report the optimal threshold
51                 print(f"\nBest Threshold: {best_threshold:.2f} with F1={best_f1:.2f}")
52
53     if __name__ == "__main__":
54         run_test()

```

- src/gui/

hasilPencarian.py

home.py

```
1 import flat as ft
2
3 def HomePage(page: ft.Page):
4     return ft.View(
5         route="/",
6         controls=[],
7         ft.Container(
8             expand=True,
9             padding=ft.padding.symmetric(horizontal=40, vertical=20),
10            alignment=ft.alignment.center,
11            content=ft.Column(
12                expand=True,
13                spacing=0,
14                alignment=ft.MainAxisAlignment.START,
15                controls=[
16                    # Navigation bar
17                    ft.Container(
18                        padding=ft.padding.symmetric(vertical=40),
19                        alignment=ft.alignment.top_center,
20                        content=ft.Column(
21                            alignment=ft.MainAxisAlignment.CENTER,
22                            spacing=20,
23                            controls=[
24                                ft.textButton("Beranda", style=ft.ButtonStyle(text_size=20, decoration=ft.TextDecoration.UNDERLINE)), on_click=lambda _: page.go("") ,
25                                ft.textButton("Pencarian", style=ft.ButtonStyle(text_size=20, decoration=ft.TextDecoration.UNDERLINE)), on_click=lambda _: page.go("/pencarian"),
26                                ft.textButton("Tentang Kami", style=ft.ButtonStyle(text_size=20, decoration=ft.TextDecoration.UNDERLINE)), on_click=lambda _: page.go("/tentang")
27                            ]
28                        )
29                    ),
30
31                    # Spacer atas
32                    ft.Container(expand=1),
33
34                    # Konten utama
35                    ft.Container(
36                        alignment=ft.alignment.center,
37                        content=ft.Column(
38                            alignment=ft.Alignment.CENTER,
39                            spacing=0,
40                            horizontal_alignment=ft.CrossAxisAlignment.CENTER,
41                            controls=[
42                                ft.text(
43                                    "Yuk, Review CV Aku Dong! 📄",
44                                    size=14,
45                                    weight=ft.FontWeight.BOLD,
46                                    text_align=ft.TextAlign.CENTER,
47                                    font_family="OSO-Regular"
48                                ),
49                                ft.Container(
50                                    margin=ft.margin.symmetric(vertical=30),
51                                    content=ft.Text(
52                                        "Punten Kak HHHH, maaf banget tapi aku lagi cari staff baru nih...\nini requirements-nya ya, Kak. Nanti aku izin catch up lagi. HANUPIS!",
53                                        size=20,
54                                        text_align=ft.TextAlign.CENTER,
55                                        font_family="OSO-Regular"
56                                    )
57                                ),
58                                ft.textButton(
59                                    on_click=lambda _: page.go("/pencarian"),
60                                    style=ft.ButtonStyle(
61                                        color="#3661BE",
62                                        padding=ft.padding.symmetric(horizontal=50, vertical=20),
63                                        shape=ft.RoundedRectangleBorder(radius=50),
64                                        elevation=3,
65                                        side=ft.BorderSide(1, "#3661BE"),
66                                    ),
67                                    content=ft.Text(
68                                        "Wah! Carl Yuk, Kak!",
69                                        size=20,
70                                        color="#3661BE"
71                                    )
72                                )
73                            ]
74                        )
75                    ),
76
77                    # Spacer bawah
78                    ft.Container(expand=1),
79
80                ],
81            ],
82            padding=0,
83            spacing=0
84        )
85    )
```

pencarian.py

tentang.py

[summary.py](#)

```
 1 import flet as ft
 2
 3 def SummaryPage(data: dict, page: ft.Page):
 4     def section(title: str, content: str):
 5         return ft.Column(
 6             spacing=5,
 7             controls=[
 8                 ft.Text(title, size=20, weight=ft.FontWeight.BOLD),
 9                 ft.Text(content if content else "-", size=15, text_align=ft.TextAlign.JUSTIFY)
10             ]
11         )
12
13     def handle_back():
14         if page.session.contains_key("summary_data"):
15             page.session.remove("summary_data")
16
17         search_results = page.session.get("search_results")
18         search_timings = page.session.get("search_timings")
19
20         if search_results is not None and search_timings is not None:
21             from ..ui.hasilPencarian import ResultPage
22             page.views.append(ResultPage(search_results, search_timings, page))
23             page.update()
24         else:
25             page.go("/pencarian")
26
27     # Navigation bar
28     navbar = ft.Container(
29         padding=ft.padding.symmetric(vertical=40),
30         alignment=ft.alignment.top_center,
31         content=ft.Row(
32             alignment=ft.MainAxisAlignment.CENTER,
33             spacing=10,
34             controls=[
35                 ft.TextButton("Beranda", style=ft.ButtonStyle(text_size=20, decoration=ft.TextDecoration.UNDERLINE), on_click=lambda _: page.go("/")),
36                 ft.TextButton("Pencarian", style=ft.ButtonStyle(text_size=20, decoration=ft.TextDecoration.UNDERLINE), on_click=lambda _: page.go("/pencarian")),
37                 ft.TextButton("Tentang Kru", style=ft.ButtonStyle(text_size=20, decoration=ft.TextDecoration.UNDERLINE), on_click=lambda _: page.go("/tentang"))
38             ]
39         )
40     )
41
42     # Tombol back
43     back_button = ft.Container(
44         alignment=ft.alignment.center_left,
45         content=ft.TextButton(
46             on_click=handle_back,
47             style=ft.buttonstyle(
48                 bgcolor="#FFFFFF",
49                 padding=ft.padding.symmetric(horizontal=20, vertical=10),
50                 shape=ft.RoundedRectangleBorder(radius=30),
51                 elevation=2,
52                 side=ft.BorderSide(1, "#36618E"),
53             ),
54             content=ft.Row(
55                 spacing=5,
56                 tight=True,
57                 controls=[
58                     ft.Text("Kembali", size=20, color="#36618E")
59                 ]
60             )
61         )
62     )
63
64     # Konten card utama
65     center_card = ft.Container(
66         width=1000,
67         padding=30,
68         border_radius=20,
69         border=ft.border.all(1),
70         bgcolor="#EBEDF6",
71         content=ft.Column(
72             spacing=25,
73             controls=[
74                 section("Summary", data.get("summary", "")),
75                 section("Skills", data.get("skills", "")),
76                 section("Education", data.get("education", "")),
77                 section("Experience", data.get("experience", "")),
78             ]
79         )
80     )
81
82     return ft.view(
83         route="/summary",
84         controls=[
85             ft.Container(
86                 expand=True,
87                 padding=ft.padding.symmetric(vertical=40, horizontal=20),
88                 content=ft.Column(
89                     horizontal_alignment=ft.CrossAxisAlignment.CENTER,
90                     expand=True,
91                     scroll=ft.ScrollMode.AUTO,
92                     spacing=15,
93                     controls=[
94                         navbar,
95                         ft.Container(
96                             width=1000,
97                             padding=ft.padding.symmetric(horizontal=20),
98                             content=ft.Row(
99                                 alignment=ft.MainAxisAlignment.SPACE_BETWEEN,
100                                 spacing=1,
101                                 controls=[
102                                     ft.Text("Ringkasan CV 📄", size=40, weight=ft.FontWeight.BOLD),
103                                     back_button
104                                 ]
105                             ),
106                         ),
107                         center_card
108                     ]
109                 )
110             ]
111         )
112     )
113 
```

4.2. Penjelasan Tata Cara Penggunaan Program

Setup Program:

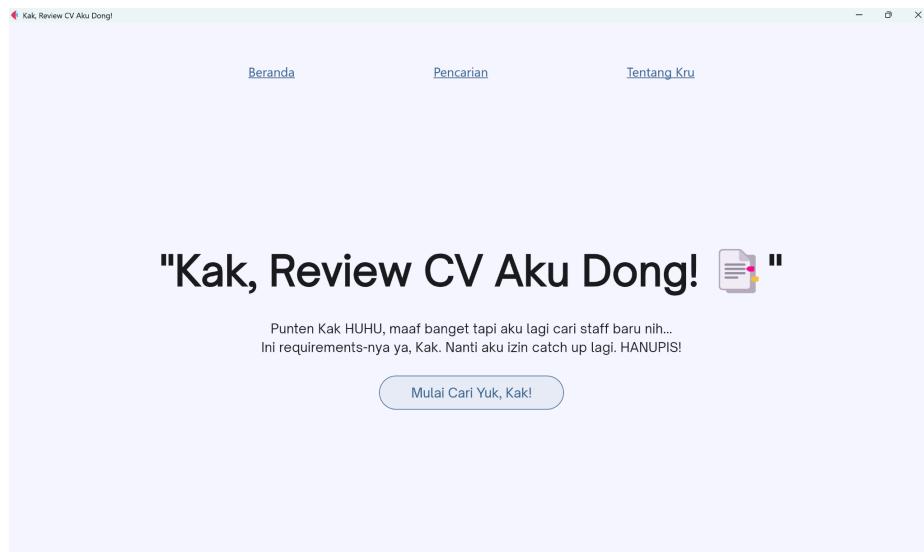
1. Klon pranala repository yang terlampir
2. Pindah ke direktori src pada terminal
3. Buat dan aktivasi python virtual environment
4. Install package yang diperlukan untuk menjalankan program
5. Konfigurasi koneksi database lokal, buka file .env yang baru dibuat di editor teks dan masukkan password MySQL lokal untuk variabel ATS_DB_PASS.
6. Setup database di direktori src/
7. Setelah database berhasil diinisialisasi, di dalam direktori src/, jalankan main.py untuk membuat tabel, mengisi data awal di terminal, dan memberi peringkat CV dari folder data/.

Cara Penggunaan program:

1. Saat menjalankan main.py akan ditampilkan antarmuka Home Page, pengguna bisa menekan tombol pada halaman utama, atau menekan tombol pencarian untuk melakukan pencarian.
2. Dalam halaman pencarian, masukkan kata kunci yang ingin dicari dalam CV, lalu memasukkan banyak applicant yang ingin dicari, lalu algoritma yang ingin digunakan.
3. Jika sudah mengisi parameter yang digunakan, klik tombol “cari sekarang!” untuk melihat hasil
4. Applicant akan muncul berdasarkan peringkat seberapa banyak kata kunci yang ingin dicari ada dalam cv.
5. Pengguna juga bisa melihat summary dan file .pdf berisi CV dari halaman hasil pencarian.

Interface Program:

1. Home Page: merupakan halaman utama dan bisa navigasi ke halaman pencarian untuk melakukan pencarian



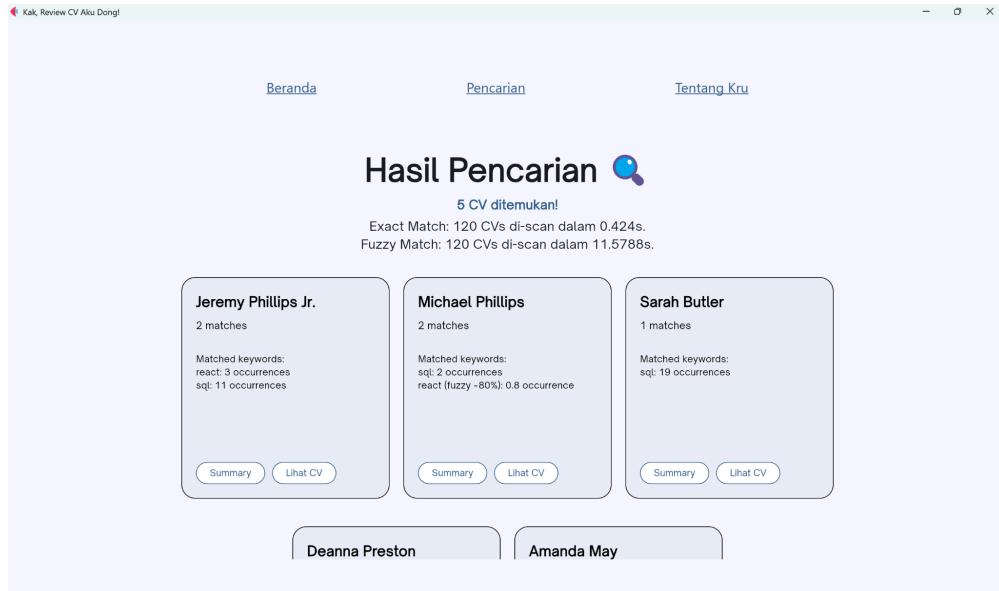
Tampilan Awal

2. Halaman Pencarian: Halaman untuk pengguna memasukkan kata kunci, banyak applicant yang ingin dilihat, serta algoritma yang ingin digunakan

The screenshot shows the search page of the website. The title "Yuk Cari Kandidat Terbaik, Kak!" is displayed with a person icon. Below the title, there is a placeholder text: "Masukkan beberapa informasi berikut untuk mencari kandidat.". The search form consists of three input fields: "Kata Kunci*" (with a placeholder "Contoh: python, react, sql"), "Jumlah Hasil*" (with a placeholder "Contoh: 12"), and "Algoritma Pencarian*" (a dropdown menu with the option "Pilih Algoritma"). At the bottom of the form is a blue button labeled "Cari Sekarang!".

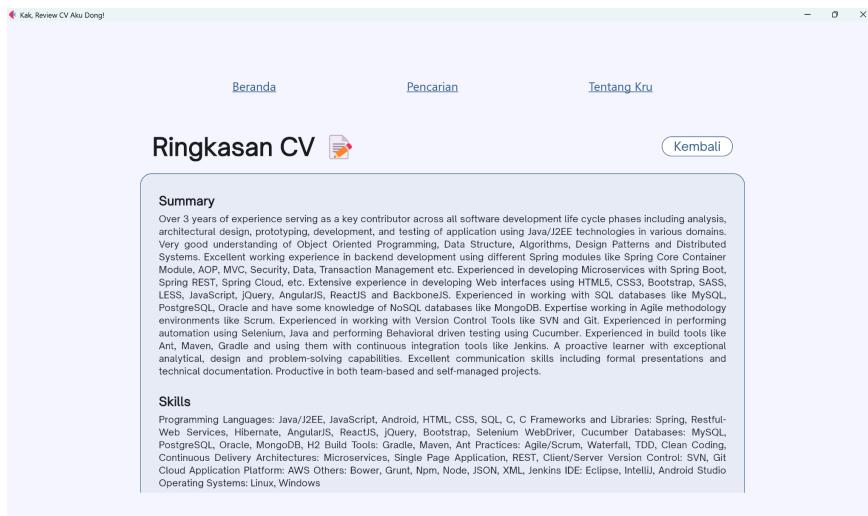
Tampilan Pencarian

3. Hasil Pencarian: Halaman yang menampilkan hasil pencarian, untuk melihat cv, serta tombol untuk melihat summary cv



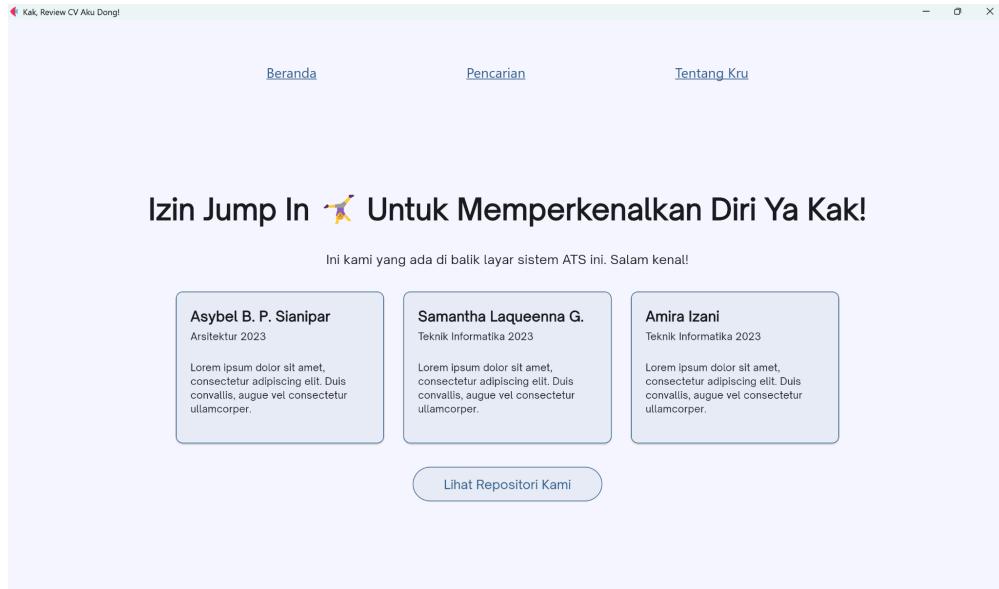
Tampilan Hasil Pencarian

- Summary Page: Halaman untuk melihat summary dari hasil pencarian seorang applicant



Tampilan Summary

- Tentang Kru Page



Tampilan Tentang Kru

4.3. Hasil dan Analisis Pengujian Parameter Algoritma

4.3.1. Pengujian Parameter

Sebelum melakukan pengujian fungsionalitas aplikasi secara menyeluruh, perlu ditentukan beberapa parameter konfigurasi yang optimal untuk algoritma yang digunakan agar sistem dapat berjalan dengan efektif. Salah satu parameter yang paling krusial dalam sistem ini adalah nilai ambang batas kemiripan (*similarity threshold*) untuk algoritma Levenshtein Distance pada fitur *fuzzy matching*.

Tujuan dari pengujian parameter ini adalah untuk menemukan nilai threshold "sweet spot" yang mampu menyeimbangkan fleksibilitas dan ketepatan kalkulasi kemiripan.

Untuk menemukan nilai threshold yang optimal, kami melakukan pengujian terpisah dengan metodologi sebagai berikut.

- **Pembuatan Dataset Uji:** Kami menyusun sebuah dataset uji yang berisi beberapa kata kunci (*target words*) yang relevan dengan domain rekrutmen, beserta variasinya. Setiap variasi diberi label sebagai **true** (seharusnya cocok) atau **false** (seharusnya tidak cocok).
- **Rentang Pengujian:** Pengujian dilakukan dengan beberapa nilai threshold yang berbeda, yaitu dari 0.70 hingga 0.95.

Metrik Evaluasi: Untuk setiap nilai threshold, kami mengukur performanya menggunakan tiga metrik standar dalam evaluasi klasifikasi:

- **Precision:** Mengukur seberapa akurat hasil pencocokan yang ditemukan. ($TP / (TP + FP)$)
- **Recall:** Mengukur seberapa lengkap sistem dalam menemukan semua kecocokan yang seharusnya ditemukan. ($TP / (TP + FN)$)
- **F1-Score:** Merupakan rata-rata harmonik dari Precision dan Recall, yang menjadi metrik utama kami untuk menemukan keseimbangan terbaik antara keduanya. (2 * (Precision * Recall) / (Precision + Recall))

Pengujian dilakukan terhadap dataset uji yang telah disiapkan. Hasil dari pengujian parameter threshold Levenshtein Distance disajikan di bawah ini.

Threshold 0.70: TP=29, FP=0, FN=8, TN=30, Precision=1.00, Recall=0.78, F1=0.88
Threshold 0.75: TP=29, FP=0, FN=8, TN=30, Precision=1.00, Recall=0.78, F1=0.88
Threshold 0.80: TP=26, FP=0, FN=11, TN=30, Precision=1.00, Recall=0.70, F1=0.83
Threshold 0.85: TP=23, FP=0, FN=14, TN=30, Precision=1.00, Recall=0.62, F1=0.77
Threshold 0.90: TP=11, FP=0, FN=26, TN=30, Precision=1.00, Recall=0.30, F1=0.46
Threshold 0.95: TP=10, FP=0, FN=27, TN=30, Precision=1.00, Recall=0.27, F1=0.43

Best Threshold: 0.75 with F1=0.88

4.3.1. Analisis Hasil Pengujian Parameter

Berdasarkan hasil pengujian parameter, dapat ditarik beberapa analisis:

- **Precision** bernilai sempurna (1.00) pada sebagian besar threshold yang relevan. Hal ini menunjukkan bahwa sistem sangat baik dalam menghindari *false positive* atau tidak salah mencocokkan kata yang sama sekali berbeda.
- **Recall** menunjukkan tren penurunan seiring dengan meningkatnya nilai threshold. Hal ini logis, karena semakin ketat aturan kemiripan (threshold tinggi), semakin banyak variasi typo yang tidak dianggap sebagai kecocokan (*false negative*).
- **F1-Score** sebagai metrik penyeimbang mencapai nilai puncaknya, yaitu **0.88**, pada rentang threshold 0.70 dan 0.75. Ini menandakan bahwa keseimbangan terbaik antara Precision dan Recall berada pada rentang tersebut.

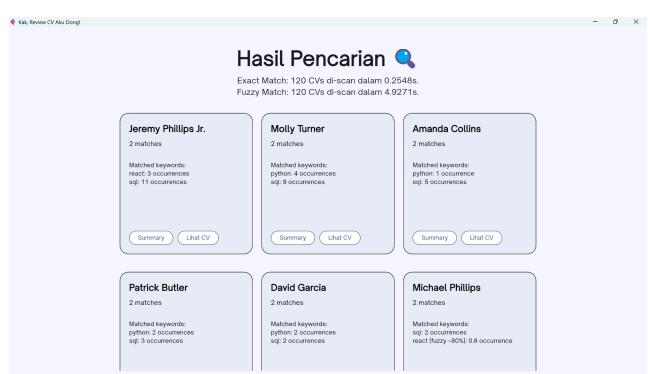
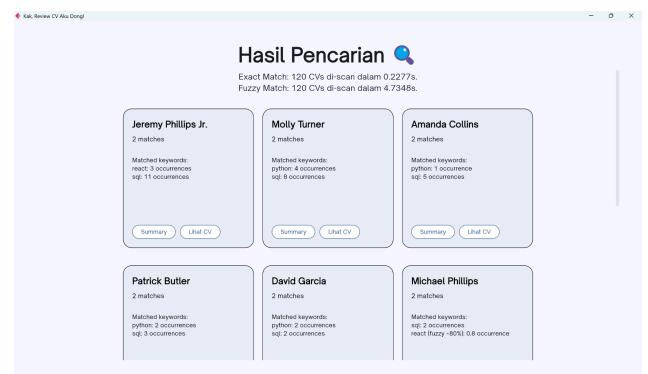
Ketika terdapat beberapa nilai threshold yang menghasilkan F1-Score maksimal yang sama, praktik terbaik adalah memilih nilai **tertinggi (paling ketat)** dari rentang tersebut.

Hal ini dilakukan untuk meminimalkan risiko *false positive* pada data yang lebih beragam, sambil tetap mempertahankan tingkat *recall* yang optimal.

Oleh karena itu, berdasarkan analisis data di atas, kami menetapkan nilai ambang batas kemiripan (*similarity threshold*) untuk Levenshtein Distance sebesar **0.75** untuk digunakan pada aplikasi ini.

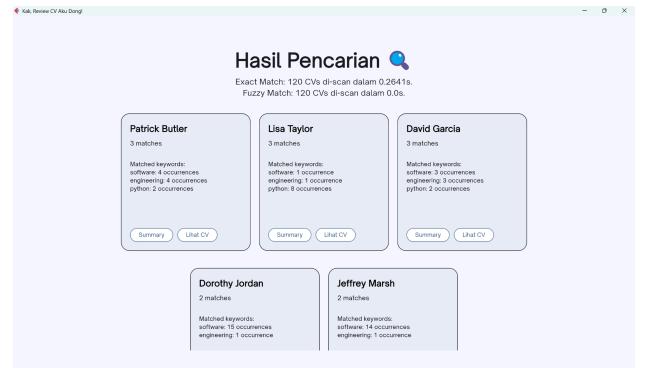
4.4. Hasil dan Analisis Pengujian Aplikasi ATS

4.4.1 Hasil Pengujian Aplikasi ATS

Uji Kasus	Tampilan Hasil								
1. Pengujian Menggunakan Algoritma KMP dan BM									
Parameter yang digunakan: - Kata kunci: python, sql, react - Jumlah Hasil: 10 - Algoritma: KMP	 <p>The screenshot shows a search interface titled 'Hasil Pencarian'. It displays 10 search results in a grid format. Each result card includes the name, number of matches, and the specific keywords found. Buttons for 'Summary' and 'Lihat CV' are visible at the bottom of each card.</p> <table border="1"> <thead> <tr> <th>Matched Keywords</th> <th>Count</th> </tr> </thead> <tbody> <tr> <td>python</td> <td>2 occurrences</td> </tr> <tr> <td>sql</td> <td>3 occurrences</td> </tr> <tr> <td>react</td> <td>1 occurrence</td> </tr> </tbody> </table>	Matched Keywords	Count	python	2 occurrences	sql	3 occurrences	react	1 occurrence
Matched Keywords	Count								
python	2 occurrences								
sql	3 occurrences								
react	1 occurrence								
Parameter yang digunakan: - Kata kunci: python, sql, react - Jumlah Hasil: 10 - Algoritma: BM	 <p>The screenshot shows a search interface titled 'Hasil Pencarian'. It displays 10 search results in a grid format. Each result card includes the name, number of matches, and the specific keywords found. Buttons for 'Summary' and 'Lihat CV' are visible at the bottom of each card.</p> <table border="1"> <thead> <tr> <th>Matched Keywords</th> <th>Count</th> </tr> </thead> <tbody> <tr> <td>python</td> <td>2 occurrences</td> </tr> <tr> <td>sql</td> <td>3 occurrences</td> </tr> <tr> <td>react</td> <td>1 occurrence</td> </tr> </tbody> </table>	Matched Keywords	Count	python	2 occurrences	sql	3 occurrences	react	1 occurrence
Matched Keywords	Count								
python	2 occurrences								
sql	3 occurrences								
react	1 occurrence								

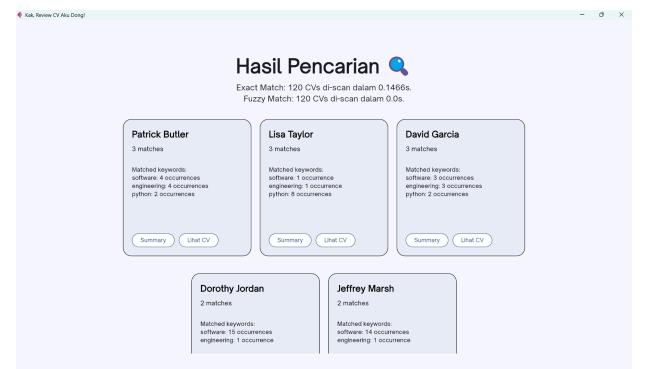
Parameter yang digunakan:

- Kata kunci: engineering, python, software
- Jumlah Hasil: 5
- Algoritma: KMP



Parameter yang digunakan:

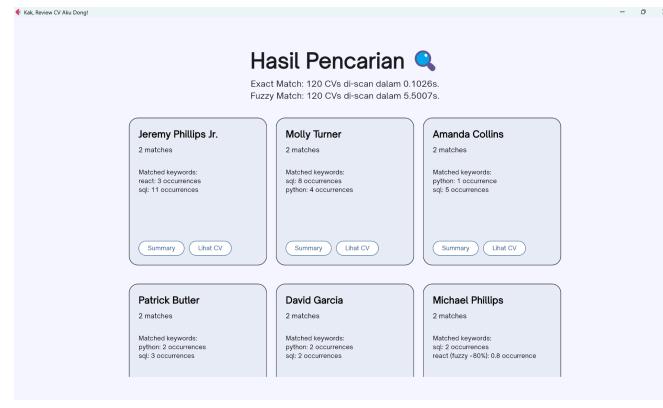
- Kata kunci: engineering, python, software
- Jumlah Hasil: 5
- Algoritma: BM



2. Pengujian Menggunakan Algoritma Aho-Corasick

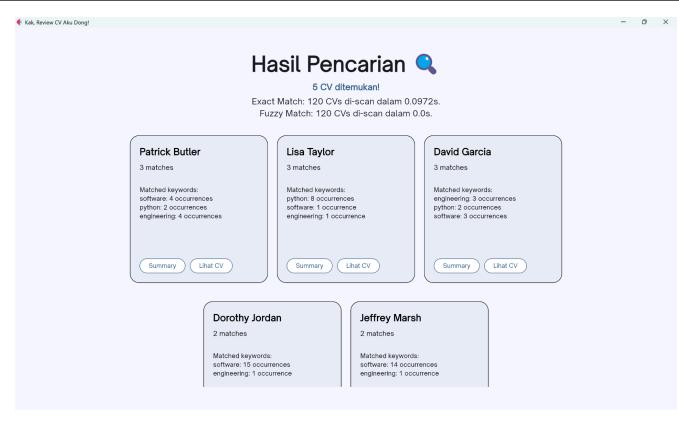
Parameter yang digunakan:

- Kata kunci: python, sql, react
- Jumlah Hasil: 10
- Algoritma: AC



Parameter yang digunakan:

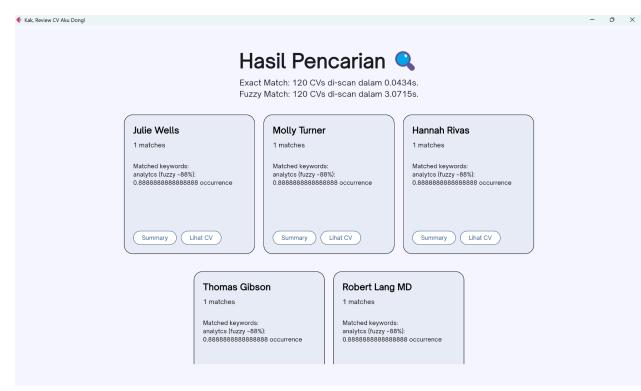
- Kata kunci: engineering, python, software
- Jumlah Hasil: 5
- Algoritma: AC



3. Pengujian metrik Levenshtein Distance dalam proses pencocokan

Parameter yang digunakan:

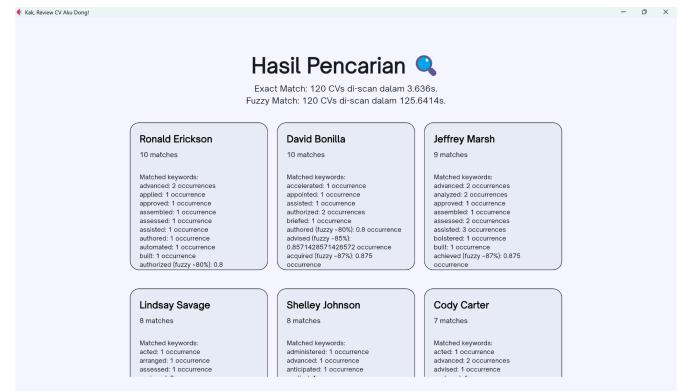
- Kata kunci: analytics
- Jumlah Hasil: 5
- Algoritma: BM

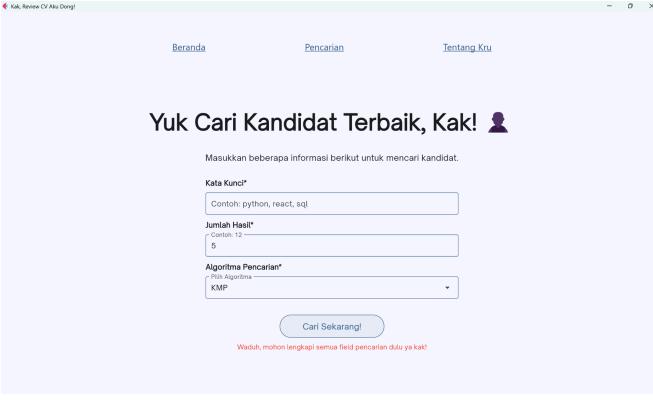
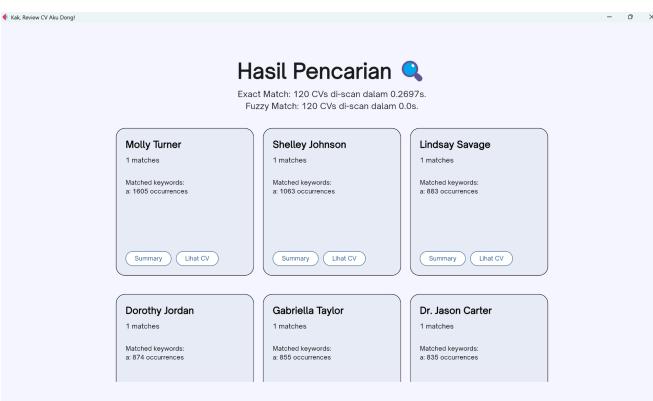
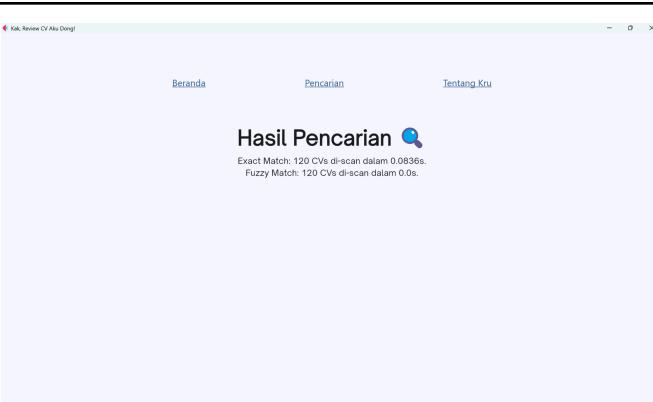


4. Pengujian Performa Algoritma dengan input banyak kata kunci

Parameter yang digunakan:

- Kata kunci:
- Accelerated, Achieved, Acquired, Acted, Adapted, Addressed, Administered, Advanced, Advised, Advocated, Aided, Aligned, Allocated, Analyzed, Anticipated, Applied, Appointed, Appraised, Approved, Arbitrated, Arranged, Assembled, Assessed, Assigned, Assisted, Attained, Audited, Augmented, Authored, Authorized, Automated, Awarded, Balanced, Benchmarked, Bolstered, Boosted, Brainstormed, Briefed, Broadened, Budgeted, Built, Calculated, Calibrated



<ul style="list-style-type: none"> - Jumlah Hasil: 500 - Algoritma: KMP 	
<h2>5. Pengujian Validasi Input Kata Kunci</h2>	
<p>Parameter yang digunakan:</p> <ul style="list-style-type: none"> - Kata kunci: "" (kosong) - Jumlah Hasil: 5 - Algoritma: BM 	
<p>Parameter yang digunakan:</p> <ul style="list-style-type: none"> - Kata kunci: a - Jumlah Hasil: 10 - Algoritma: BM 	
<p>Parameter yang digunakan:</p> <ul style="list-style-type: none"> - Kata kunci: nonexistentkeyword123 - Jumlah Hasil: 5 - Algoritma: KMP 	

4.4.2 Analisis Pengujian Aplikasi ATS

Berdasarkan Hasil Uji Kasus, terlihat bahwa pencarian exact match dengan kata kunci yang sama, Algoritma dengan waktu eksekusi tercepat adalah Aho-Corasick, diikuti dengan Algoritma Boyer-Moore, lalu KMP. Hal ini sesuai dengan teori yang digunakan.

Pada pengujian fuzzy match menggunakan levenshtein Distance, juga berfungsi dengan benar dengan uji nilai dari rentang 0.0 hingga 1.0. Hal ini membuktikan bahwa implementasi Levenshtein Distance beserta dengan parameter threshold yang telah divalidasi mampu memenuhi tujuannya, yaitu memberikan toleransi terhadap kesalahan pengetikan minor tanpa mengorbankan relevansi hasil pencarian secara signifikan.

4.5. Implementasi Bonus

4.5.1. Aho-Corasick

Algoritma Aho-Corasick, yang teori lengkapnya telah dibahas pada Bab 3, diimplementasikan sebagai opsi ketiga untuk pencocokan kata kunci tepat (exact matching). Implementasi ini bertujuan untuk memberikan alternatif pencarian yang sangat efisien ketika pengguna memasukkan beberapa kata kunci sekaligus.

Logika algoritma Aho-Corasick dibungkus secara *object-oriented* dalam sebuah kelas `AhoCorasick` yang ditempatkan pada modul `src/algorithms/aho_corasick.py`. Pendekatan berbasis kelas ini dipilih untuk mengelola struktur data *Trie* dan *failure links* yang kompleks secara bersih dan terstruktur.

- **Pra-pemrosesan:** Proses pra-pemrosesan yang terdiri dari pembuatan *Trie* (melalui metode `_build_trie()`) dan pembangunan *failure links* (melalui metode `_build_failure_links()`) dieksekusi secara otomatis di dalam konstruktor (`__init__`) kelas saat objek `AhoCorasick` dibuat dengan daftar kata kunci.
- **Metode `search(text)`:** Kelas ini menyediakan satu metode publik utama, yaitu `search()`, yang menerima teks CV sebagai input. Metode ini melakukan pemindaian teks dalam satu kali lintasan dan mengembalikan sebuah kamus (*dictionary*) yang berisi semua kata kunci yang ditemukan beserta daftar indeks kemunculannya.

Integrasi Aho-Corasick ke dalam alur kerja utama aplikasi dilakukan di dalam modul `search_engine.py`.

- Aho-Corasick ditambahkan sebagai opsi ketiga (AC) yang dapat dipilih oleh pengguna melalui antarmuka.

- Jika pengguna memilih "AC", `search_engine.py` akan membuat satu objek `AhoCorasick` dengan semua kata kunci yang dimasukkan pengguna. Objek ini kemudian digunakan untuk memanggil metode `search()` satu kali untuk setiap CV yang diproses.
- Struktur hasil yang dikembalikan oleh metode `search()` telah disesuaikan agar konsisten dengan format yang dihasilkan oleh algoritma KMP dan Boyer-Moore, sehingga sisa alur kerja seperti penilaian (*scoring*) dan pemeringkatan tidak memerlukan perubahan.

4.5.2. Enkripsi

Untuk memenuhi bonus keamanan data, kami mengimplementasikan fitur enkripsi untuk melindungi data pribadi pelamar (*Personally Identifiable Information* - PII) yang tersimpan di dalam basis data.

Tujuan dari enkripsi ini adalah untuk mengamankan data *at-rest*. Artinya, jika seseorang berhasil mendapatkan akses langsung ke file basis data, mereka tidak akan dapat membaca informasi sensitif karena data tersebut disimpan dalam format terenkripsi.

Sesuai dengan batasan pada spesifikasi tugas, kami mengimplementasikan Vigenère Cipher secara manual. Alasan pemilihan Vigenère Cipher adalah karena tingkat keamanannya yang memadai untuk lingkup tugas ini (lebih kuat dari Caesar Cipher) dan kompleksitasnya yang realistik untuk diimplementasikan dari awal tanpa pustaka eksternal.

Logika enkripsi dienkapsulasi dalam modul utilitas `src/utils/encryption.py`.

- Fungsi `encrypt()` dan `decrypt()`: Modul ini menyediakan dua fungsi utama. Fungsi `encrypt` mengenkripsi teks menggunakan Vigenère Cipher, lalu mengkodekannya ke format Base64 agar aman disimpan di basis data. Fungsi `decrypt` melakukan proses sebaliknya.
- Manajemen Kunci: Kunci enkripsi (`SECRET_KEY`) diambil dari file `.env` untuk memastikan kerahasiaannya dan tidak disimpan langsung di dalam kode.

Enkripsi diintegrasikan secara transparan pada lapisan model (`src/model/models.py`). Pada `insert_applicant_profile`, sebelum data disimpan, setiap kolom yang berisi PII dilewatkan melalui fungsi `encrypt()`. Pada `fetch_applicant_by_id`, setelah data diambil, setiap kolom yang terenkripsi dilewatkan melalui fungsi `decrypt()` sebelum dikembalikan

ke bagian lain dari aplikasi. Dengan pendekatan ini, seluruh aplikasi bekerja dengan data yang dapat dibaca, sementara data yang tersimpan di basis data MySQL tetap aman.

BAB V – KESIMPULAN, SARAN, DAN REFLEKSI

5.1. Kesimpulan

Melalui Tugas Besar Strategi Algoritma ini, kami diminta untuk membangun sistem ATS (Applicant Tracking System) yang mampu melakukan pencarian dan pencocokan informasi dari dokumen CV digital menggunakan algoritma pencocokan string. Kami mengembangkan aplikasi desktop dengan antarmuka pengguna berbasis Flet dan berbahasa Python, serta menerapkan algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan metrik Levenshtein Distance dalam proses pencocokan.

Berdasarkan pengujian yang kami lakukan, algoritma BM menunjukkan performa yang relatif lebih cepat dibandingkan algoritma KMP dalam proses pencarian kata kunci. Hal ini disebabkan oleh kemampuannya dalam melakukan lompatan yang lebih besar saat mismatch terjadi. Pada teks panjang seperti CV, algoritma BM menjadikan proses pencarian relatif lebih efisien. Sementara, algoritma Aho-Corasick cocok untuk digunakan untuk kata kunci yang lebih banyak.

Dalam kasus ketika sistem tidak menemukan pencocokan exact match, kami menggunakan algoritma Levenshtein Distance untuk melakukan pencarian fuzzy match yang tetap relevan meskipun terdapat kesalahan pengetikan atau variasi kata (typo). Proses ini membuktikan bahwa pengguna bisa mendapatkan hasil pencarian yang mendekati harapannya, meskipun input kurang sesuai.

Penggunaan regex dalam sistem ini mengekstraksi informasi penting dari CV seperti nama, keahlian, dan riwayat pendidikan secara otomatis. Secara keseluruhan sistem yang telah kami bangun dapat berjalan dengan baik, mampu menangani variasi data, dan memberikan hasil pencarian yang sesuai dalam waktu yang relatif singkat.

5.2. Saran

Proses pengembangan sistem ATS ini memiliki kekurangan. Berikut merupakan saran yang dapat kami berikan bagi pembaca atau pengembang lainnya:

1. Membaca Dokumentasi

Untuk membangun aplikasi menggunakan framework dan bahasa tertentu, penting bagi pengembang untuk terlebih dahulu membaca dokumentasi resmi sebelum mempelajari contoh-contoh kode program. Dengan memahami

dokumentasi secara menyeluruh, pengembang dapat menggunakan fitur yang tersedia secara optimal, menyesuaikan dengan kebutuhan proyek, dan menghindari praktik-praktik yang tidak direkomendasikan. Setelah pemahaman dasar terbentuk, barulah contoh kode menjadi lebih bermakna dan efektif sebagai acuan implementasi.

2. Kerja Sama Kelompok

Pengerjaan tugas akan lebih mudah dilalui jika kelompok telah membuat strategi terlebih dahulu. Strategi yang dimaksud dapat berupa pembagian tugas yang adil. Masing-masing anggota kelompok juga harus berkontribusi aktif selama pengerjaan tugas. Manajemen waktu juga merupakan aspek yang sangat penting dalam pengerjaan suatu tugas mandiri maupun berkelompok. Setiap anggota kelompok harus dapat membagi waktu dengan baik dan mempunyai kedisiplinan. Jika anggota kelompok sedang terkendala waktu dikarenakan kesibukan lainnya, anggota tersebut wajib mengabari kelompoknya agar kelompok dapat saling membantu.

5.3. Refleksi

Tugas besar ini memberikan kami pemahaman yang lebih mendalam terhadap penerapan algoritma pencocokan string dalam implementasi nyata seperti proses rekrutmen tenaga kerja. Kami juga kembali mempelajari bahwa efisiensi algoritma dalam mengolah data atau dokumen besar sangat penting.

Selain itu, kami juga menyadari pentingnya kolaborasi tim dalam membangun sistem yang cukup kompleks, mulai dari pemrosesan data, pembangunan algoritma, dan integrasi antarmuka dengan logika.

DAFTAR PUSTAKA

Institut Teknologi Bandung. (2024). *Spesifikasi Tugas Besar 3 Stima 2024/2025* [Google Document]. Diambil dari

<https://docs.google.com/document/d/1aVJtQ9oj-WE7GrQwaHv3xzYcAMIsplSBLAX4SLBcPZQA/edit?tab=t.0>

Munir, R. (2025). Strategi Algoritma 2024/2025. Diambil dari

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/stima24-25.htm>

LAMPIRAN

A. Pranala Repository

https://github.com/amiraIzani/Tubes3_KakReviewCVakuDong

B. Pranala Video

<https://youtu.be/trx1D8aIufI>

C. Tabel Ketercapaian

No	Poin	Ya	Tidak
1	Aplikasi dapat dijalankan.	✓	
2	Aplikasi menggunakan basis data berbasis SQL dan berjalan dengan lancar.	✓	
3	Aplikasi dapat mengekstrak informasi penting menggunakan Regular Expression (Regex).	✓	
4	Algoritma <i>Knuth-Morris-Pratt (KMP)</i> dan <i>Boyer-Moore (BM)</i> dapat menemukan kata kunci dengan benar.	✓	
5	Algoritma Levenshtein Distance dapat mengukur kemiripan kata kunci dengan benar.	✓	
6	Aplikasi dapat menampilkan <i>summary CV applicant</i> .	✓	
7	Aplikasi dapat menampilkan <i>CV applicant</i> secara keseluruhan.	✓	

8	Membuat laporan sesuai dengan spesifikasi.	✓	
9	Membuat bonus enkripsi data profil <i>applicant</i> .	✓	
10	Membuat bonus algoritma Aho-Corasick.	✓	
11	Membuat bonus video dan diunggah pada Youtube.	✓	