

Laporan Tugas Besar 3

IF2211 Strategi Algoritma

Pemanfaatan *Pattern Matching* untuk Membangun Sistem *Applicant Tracking System* Berbasis CV Digital



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T., M.Sc.

Disusun oleh:

Kelompok 54 – HRProfesional

Maheswara Bayu Kaindra (13523015)
Mayla Yaffa Ludmilla (13523050)
Angelina Efrina Prahastaputri (13523060)

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika**

Institut Teknologi Bandung

**Jl. Ganesa 10, Bandung 40132
2025**

BAB I

Deskripsi Tugas

Batas Pengumpulan : Minggu, 15 Juni 2025, 22.11 WIB.

Arsip Pengumpulan : *Source program yang dapat dijalankan disertai README dan laporan (soft copy).*



*Gambar 1.1 : 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.

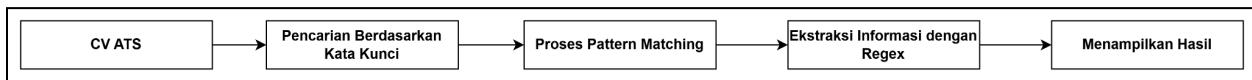
Meskipun demikian, salah satu tantangan besar dalam pengembangan sistem ATS adalah kemampuan untuk memproses dokumen CV dalam format PDF yang tidak selalu terstruktur. Dokumen seperti ini memerlukan metode canggih untuk mengekstrak informasi penting seperti identitas, pengalaman kerja, keahlian, dan 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.

1.1 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 1.2 : 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.

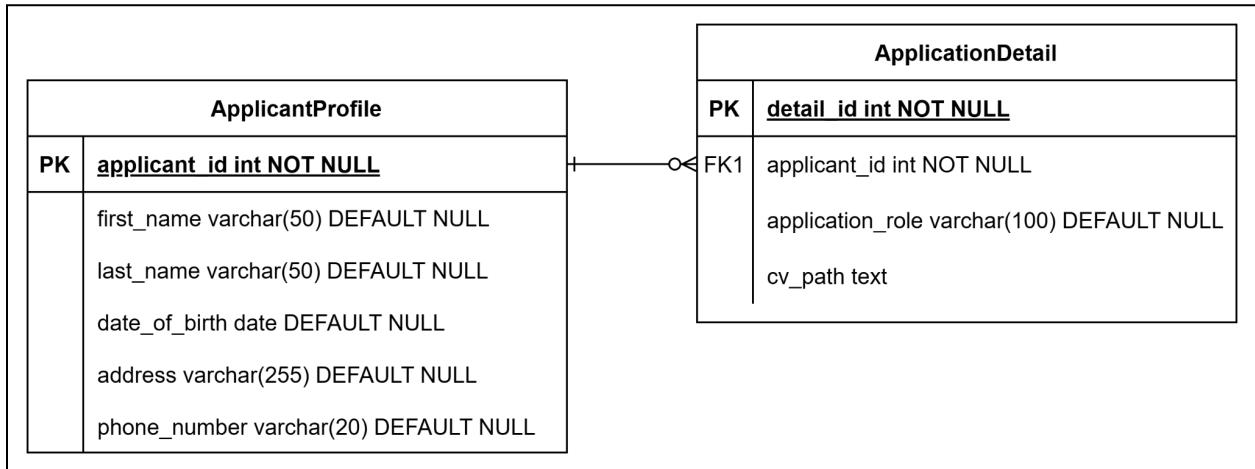
Untuk memberikan pemahaman yang lebih konkret, berikut disajikan contoh kasus penerapan sistem CV ATS beserta prosesnya dan contoh output yang dihasilkan. Dataset yang digunakan dalam contoh ini merupakan dataset CV ATS yang tercantum pada bagian referensi.

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

Tabel 1.1 : Hasil Ekstraksi teks dari CV ATS.

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 1.3 : 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.

1.2 Penggunaan Program

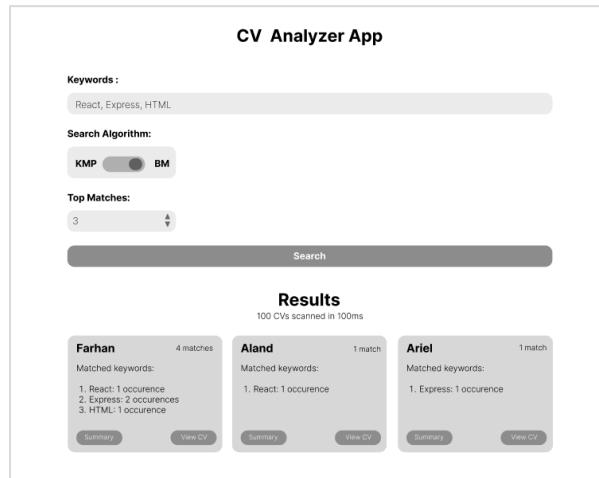
Anda diperbolehkan menambahkan elemen tambahan seperti gambar, logo, atau komponen visual lainnya. Desain antarmuka untuk aplikasi desktop **tidak wajib mengikuti tata letak persis** seperti contoh yang diberikan, namun harus dibuat semenarik mungkin, serta tetap mencakup seluruh **komponen wajib yang telah ditentukan**:

1. Judul Aplikasi
2. Kolom input data kunci memungkinkan pengguna memasukkan satu atau lebih *keyword*, yang dipisahkan dengan koma, seperti contoh: React, Express, HTML.

3. Tombol toggle memungkinkan pengguna memilih salah satu dari dua algoritma pencarian, yaitu KMP atau BM, dengan hanya satu algoritma yang bisa dipilih pada satu waktu.
4. *Top Matches Selector* digunakan untuk memilih CV teratas yang ingin ditampilkan berdasarkan hasil pencocokan.
5. *Search button* digunakan untuk memulai proses pencarian. Diletakkan secara mencolok di bawah *input field*.
6. Summary Result Section berisi informasi waktu eksekusi pencarian untuk kedua tipe matching yang dilakukan (exact match dengan KMP/BM dan fuzzy match dengan Levenshtein Distance), misalnya: “Exact Match: 100 CVs scanned in 100ms.\n Fuzzy Match: 100 CVs scanned in 101ms.”
7. Container hasil pencarian atau kartu CV digunakan untuk menampilkan data hasil pencocokan berdasarkan keyword yang sesuai. Setiap kartu memuat informasi seperti nama kandidat, jumlah kecocokan yang dihitung dari jumlah keyword yang ditemukan, serta daftar kata kunci yang cocok beserta frekuensi kemunculannya. Selain itu, tersedia dua tombol aksi: tombol Summary untuk menampilkan ekstraksi informasi dari CV, serta tombol View CV yang memungkinkan pengguna melihat langsung file CV asli.

Secara umum, berikut adalah cara umum penggunaan program:

1. Pengguna memasukkan kata kunci pencarian.
2. Memilih algoritma pencocokan: KMP atau BM.
3. Menentukan jumlah hasil yang ingin ditampilkan.
4. Menekan tombol Search.
5. Sistem menampilkan daftar CV yang paling relevan, disertai tombol untuk melihat detail (*Summary*) atau CV asli (*view CV*).



Gambar 1.4 : Contoh Antarmuka Program (Halaman Home)



Gambar 1.5 : Contoh Antarmuka Program (Halaman Summary)

1.3 Spesifikasi Wajib

Pada Tugas Besar ini, buatlah sebuah sistem yang dapat melakukan pencocokan dan pencarian informasi pelamar kerja berdasarkan dataset CV ATS Digital. Data yang digunakan merupakan gabungan **20 data pertama dari setiap category/bidang, yang telah terurut secara leksikografis** (i.e. 20 data dari category HR + 20 data dari category Designer, dst). Sistem harus memiliki fitur dengan detail sebagai berikut:

1. Sistem yang dibangun pada tugas besar ini bertujuan untuk melakukan pencocokan dan pencarian data pelamar kerja berbasis CV yang diunggah oleh pengguna. Sistem dikembangkan menggunakan **bahasa pemrograman Python** dengan **antarmuka desktop** berbasis pustaka seperti **Tkinter**, **PyQt**, atau **framework lain** yang relevan. Dalam proses pencocokan kata kunci, sistem wajib mengimplementasikan algoritma **Knuth-Morris-Pratt (KMP)** dan **Boyer-Moore (BM)**. Untuk mengukur kemiripan saat terjadi kesalahan input atau perbedaan penulisan, sistem juga menerapkan algoritma **Levenshtein Distance**. Selain itu, **Regular Expression (Regex)** digunakan untuk mengekstrak informasi penting dari teks CV secara otomatis. Oleh karena itu, penguasaan pengembangan GUI dan pemrosesan string di Python sangat penting untuk menyelesaikan tugas ini secara optimal.
2. Program yang dikembangkan harus menggunakan basis data berbasis **MySQL** untuk menyimpan informasi hasil ekstraksi dari CV yang telah diunggah. Basis data akan menyimpan informasi berupa profil pelamar beserta lokasi penyimpanan file CV di dalam sistem.
3. Fitur utama dari sistem ini adalah kemampuannya untuk ekstraksi teks dari CV dalam format PDF. Setelah dokumen CV diunggah, program harus mampu melakukan ekstraksi teks secara otomatis dan mengubahnya menjadi profil pelamar kerja. Profil tersebut akan ditampilkan kepada pengguna tanpa perlu intervensi manual tambahan. Proses ini akan membantu mempercepat identifikasi dan penilaian awal terhadap pelamar.
4. Sistem wajib menyediakan **fitur pencarian terhadap data pelamar** menggunakan **kata kunci** atau kriteria tertentu yang ditentukan oleh pengguna (misalnya nama, skill tertentu, atau

pengalaman kerja). Pencarian ini akan dilakukan terhadap semua data dalam database dan bertujuan untuk menemukan pelamar yang paling relevan dengan kriteria pencarian tersebut. Proses pencarian dilakukan sepenuhnya secara in-memory agar hasilnya cepat dan responsif. Proses pencarian utamanya dilakukan secara exact matching.

5. Setelah exact matching, apabila tidak ditemukan kecocokan secara persis, sistem harus melakukan fuzzy matching. Untuk setiap kata kunci yang tidak ditemukan satupun kemunculan saat exact matching, lakukan pencarian kembali dengan **perhitungan tingkat kemiripan menggunakan algoritma Levenshtein Distance**. Algoritma ini memungkinkan sistem untuk tetap menampilkan hasil pencarian yang relevan, meskipun terdapat perbedaan minor atau kesalahan ketik pada input pengguna. Hal ini sangat membantu pengguna untuk tetap mendapatkan hasil terbaik tanpa harus memasukkan kata kunci secara sempurna.
6. Apabila salah satu hasil pencarian di-klik, sistem harus dapat menampilkan **ringkasan/summary** dari lamaran tersebut. Pada halaman ringkasan, harus terdapat opsi (e.g. tombol) untuk **melihat CV secara keseluruhan**.
7. Informasi yang ditampilkan dalam **ringkasan/summary** CV dari hasil pencarian harus mencakup **data penting dari pelamar**, yaitu identitas (nama, kontak, dan informasi pribadi lainnya) yang diperoleh dari basis data. Kemudian terdapat beberapa data yang diperoleh dengan cara ekstraksi melalui **regular expression**, meliputi:
 - a. Ringkasan pelamar (summary/overview)
 - b. Keahlian pelamar (skill)
 - c. Pengalaman kerja (e.g tanggal dan jabatan)
 - d. Riwayat pendidikan (e.g tanggal kelulusan, universitas, dan gelar)

Dengan menampilkan informasi-informasi penting tersebut, sistem dapat memberikan ringkasan profil yang relevan kepada pengguna.

8. Pengguna aplikasi dapat memilih algoritma pencocokan yang ingin digunakan untuk *exact matching*, yaitu antara **KMP** atau **BM**, (bisa pula **Aho-Corasick** apabila mengerjakan bonus), sebelum memulai proses pencarian. Pilihan algoritma ini akan mempengaruhi cara sistem memindai dan mencocokkan kata kunci dengan isi CV. Hal ini memberikan fleksibilitas dan pemahaman algoritmik yang lebih luas bagi pengguna atau pengembang.
9. Pengguna aplikasi dapat **menentukan jumlah CV yang ditampilkan**. CV yang ditampilkan diurutkan mulai dari CV dengan jumlah kecocokan kata kunci terbanyak.
10. Setelah pencarian CV, sistem akan **menampilkan waktu pencarian**. Terdapat **2 waktu berbeda** yang perlu ditampilkan. Pertama adalah waktu pencarian ***exact match*** menggunakan algoritma KMP atau BM. Kemudian, tampilkan juga waktu pencarian ***fuzzy match*** menggunakan Levenshtein Distance apabila terdapat kata kunci yang belum ditemukan.
11. Aplikasi yang dibuat harus memiliki **antarmuka pengguna (user interface) yang intuitif dan menarik**, sehingga mudah digunakan bahkan oleh pengguna awam. Komponen-komponen penting seperti, input keyword, pemilihan algoritma, serta hasil pencarian harus disusun dengan

jelas dan rapi. Pengembang juga diperkenankan menambahkan fitur tambahan yang dapat memperkaya fungsi dan pengalaman pengguna, sebagai bentuk kreativitas dan inisiatif dalam mengembangkan sistem yang lebih bermanfaat dan inovatif.

1.3 Spesifikasi Bonus

Bagian ini hanya boleh dikerjakan apabila seluruh spesifikasi wajib dari Tugas Besar telah berhasil dipenuhi. Pengerjaan bagian bonus bersifat opsional, namun semakin banyak bagian bonus yang dikerjakan, maka semakin tinggi tambahan nilai yang bisa diperoleh.

1. Enkripsi Data Profil Applicant (**maksimal 5 poin**)

Lakukan proses enkripsi terhadap data profil applicant yang disimpan di dalam basis data untuk menjaga kerahasiaan informasi pribadi pelamar kerja. Enkripsi ini perlu diterapkan agar data tetap aman meskipun terjadi akses langsung ke basis data. Implementasi **tidak diperkenankan menggunakan pustaka enkripsi bawaan Python** (cryptography atau hashlib), **semakin kompleks dan aman skema enkripsi yang dibuat maka potensi nilai bonus pun akan semakin tinggi**.

NOTES: Data yang disediakan untuk keperluan demo tidak menggunakan enkripsi

2. Implementasi Algoritma Aho-Corasick (**maksimal 10 poin**)

Tambahkan dukungan algoritma Aho-Corasick sebagai alternatif metode pencarian kata kunci yang efisien untuk multi-pattern matching. Dengan algoritma ini, sistem dapat mencocokkan seluruh daftar keyword sekaligus dalam satu proses traversal teks, sehingga lebih cepat dibandingkan pendekatan konvensional satu per satu. Kehadiran opsi algoritma ini menunjukkan pemahaman lanjutan terhadap strategi optimasi pencarian string.

3. Pembuatan Video Aplikasi (**maksimal 5 poin**)

Buatlah video presentasi mengenai aplikasi yang telah dikembangkan, mencakup penjelasan fitur-fitur utama serta demonstrasi penggunaannya. Video harus menyertakan audio narasi dan menampilkan wajah dari seluruh anggota kelompok. Kualitas penyampaian dan visual akan mempengaruhi penilaian. Upload video ke YouTube dan pastikan video dapat diakses publik. Sebagai referensi, Anda dapat melihat video tugas besar dari tahun-tahun sebelumnya dengan kata kunci seperti “Tubes Stima”, “Tugas Besar Stima”, atau “Strategi Algoritma”.

1.4 Prosedur Pengerjaan

1. Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan “jawaban”.
2. Terdapat demo dengan asisten untuk mendemonstrasikan aplikasi yang telah dibuat. Pengumuman mengenai demo akan diberitahukan lebih lanjut oleh asisten.
3. Setiap kelompok harap mengisi nama kelompok dan anggotanya pada link berikut, paling lambat Minggu, 25 Mei 2025 pukul 22.11 WIB.  Pendataan Kelompok Tubes 3 Stima 2024/2025.

4. Diwajibkan untuk memilih asisten meskipun tidak melakukan asistensi, karena asisten yang dipilih akan menjadi asisten saat asistensi (opsional) dan demo tugas besar. Pemilihan asisten dapat dilakukan pada link berikut, paling lambat Minggu, 25 Mei 2025 pukul 22.11 WIB.
-  Pendataan Kelompok Tubes 3 Stima 2024/2025

1.5 Isi Laporan

1. **Cover:** Cover laporan ada foto anggota kelompok (foto bertiga). Foto ini menggantikan logo “gajah” ganesha.
2. **Bab 1:** Deskripsi tugas (dapat menyalin spesifikasi tugas ini).
3. **Bab 2:** Landasan Teori
 - a. Dasar teori (penjelasan algoritma Knuth-Morris-Pratt dan Boyer-Moore secara umum serta Aho–Corasick jika mengerjakan bonus).
 - b. Penjelasan singkat mengenai aplikasi web yang dibangun.
4. **Bab 3:** Analisis Pemecahan Masalah
 - a. Langkah-langkah pemecahan masalah.
 - b. Proses pemetaan masalah menjadi elemen-elemen algoritma KMP dan BM.
 - c. Fitur fungsional dan arsitektur aplikasi yang dibangun.
 - d. Contoh ilustrasi kasus.
5. **Bab 4:** Implementasi dan pengujian
 - a. Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun).
 - b. Penjelasan tata cara penggunaan program (*interface* program, fitur-fitur yang disediakan program, dan sebagainya).
 - c. Hasil pengujian **minimal 4** pencarian dengan variasi keyword, algoritma, dan panjang keyword yang berbeda-beda.
 - d. Analisis hasil pengujian.
6. **Bab 5:** Kesimpulan, saran, dan refleksi tentang tugas besar 3.
7. **Lampiran:** Tautan *repository* GitHub (dan video jika membuat).
8. **Daftar Pustaka.**

1.6 Pengumpulan Tugas

1. Program disimpan dalam repository yang bernama **Tubes3_NamaKelompok** dengan nama kelompok sesuai dengan yang di *sheets* diatas. Berikut merupakan struktur dari isi repository tersebut:
 - a. Folder src berisi **program yang dapat dijalankan**
 - b. Folder doc berisi laporan tugas besar dengan format **NamaKelompok.pdf**
 - c. Folder data berisi dataset yang digunakan untuk pengujian (folder ini juga yang akan menjadi lokasi CV ATS Digital pada saat demo)
 - d. README untuk tata cara penggunaan yang minimal berisi:
 - i. Penjelasan singkat algoritma KMP dan BM yang diimplementasikan
 - ii. Requirement program dan instalasi tertentu bila ada
 - iii. Command atau langkah-langkah dalam meng-compile atau build program
 - iv. Author (identitas pembuat)

2. Sangat disarankan untuk menggunakan ***semantic commit***. Buatlah release dengan format **v1.x** dengan x adalah nomor revisi dimulai dari revisi 0. Contoh v1.0 untuk release pertama, v1.1 untuk revisi selanjutnya.
3. Pastikan untuk membuat repository bersifat **Public** paling lambat **H+1 deadline (1 hari setelah deadline)**. Sebelum deadline repository harus bersifat **Private**.
4. Laporan dikumpulkan hari **Minggu, 15 Juni 2025** pada alamat Google Form berikut paling lambat **pukul 22.11 WIB**:
<https://bit.ly/tubes3stima25>
PERINGATAN: Keterlambatan akan mengurangi nilai sebanyak 1 poin untuk setiap menit keterlambatan.

Adapun pertanyaan terkait tugas besar ini bisa disampaikan melalui QnA berikut:

<https://bit.ly/QnA-Stima-25>.

1.7 Penilaian

1. **Bagian 1: Laporan (40%)**
 - a. Langkah-langkah pemecahan masalah, proses pemetaan masalah, fitur fungsional dan arsitektur aplikasi web yang dibangun, dan contoh ilustrasi kasus. (20 %)
 - b. Spesifikasi teknis program (struktur data, fungsi, dan prosedur yang dibangun), penjelasan tata cara penggunaan program (interface program, fitur-fitur yang disediakan program, dan sebagainya), hasil pengujian (screenshot antarmuka dan variasi pengujian kasus berdasarkan fitur aplikasi), dan analisis hasil pengujian. (15 %)
 - c. Kelengkapan komponen-komponen pada laporan dan README. (5 %)
2. **Bagian 2: Implementasi Program dan Demo (60%)**
 - a. Kecocokan output dari test case yang dimiliki Asisten. (25 %)
 - b. Kebenaran algoritma *KMP*, *BM*, *Regex*, dan *Levenshtein Distance*. (15%)
 - c. Pemahaman tugas besar dan algoritma yang telah dibuat oleh masing-masing anggota. (10%)
 - d. Keberhasilan input dan output, sesuai dengan komponen-komponen yang ada pada spesifikasi. (5%)
 - e. Modularitas/keterbacaan penulisan program. (5%)
3. **Bagian 3: Komponen Bonus (Maksimal 20 Poin)**
 - a. Program dapat mengenkripsi data profil *applicant*. (**bonus** maksimal 5 poin)
 - b. Aplikasi dapat mengimplementasi Algoritma Aho-Corasick. (**bonus** maksimal 10 poin)

Bonus dalam membuat video kelompok. (bonus maksimal 5 poin).

1.8 Perhatian

- **Dilarang keras** *copy paste* program dari internet, AI, repository lain, ataupun program milik teman. Program harus dibuat sendiri, kecurangan akan diberikan sanksi berat yaitu nilai tugas menjadi **nol**.
- Pastikan program **dapat dikompilasi setidaknya** pada *windows* dan *linux*.
- Apabila program **tidak dapat dijalankan** maka tidak akan dinilai oleh asisten.
- Keterlambatan pengumpuluan akan mengurangi 1 poin untuk setiap menit keterlambatan.

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.		

BAB II

Landasan Teori

2.1 Pattern Matching

Pattern Matching merupakan proses **pencarian lokasi pertama** di mana **P (pattern)** dalam **T (text)** bersetujuan, dengan informasi:

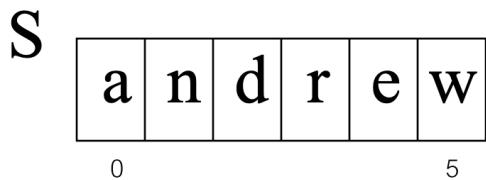
1. **T (teks / text)**, yaitu *string* berukuran sangat panjang (panjangnya n karakter).
2. **P (pattern)**, yaitu *string* dengan panjang m karakter (di mana m jauh lebih kecil dari n) yang akan dicari dalam T .

Contohnya, T = “Kami adalah HRProfesional, *developer* dari project ini.” dan P = “develop”. *Pattern Matching* seringkali digunakan dalam pencarian di berbagai media digital hingga pencarian pada *search engine*. Dalam definisi ini, *pattern matching* dibatasi untuk *string*. Nyatanya, *pattern matching* juga sering digunakan dalam analisis citra (gambar) seperti sidik jari dan mencari informasi genetik melalui pencocokan rantai asam amino.

Dalam *pattern matching* berbasis *string*, dikenal **prefix** dan **suffix**. Misalkan S merupakan string dengan ukuran m , $S = x_0x_1\dots x_{m-1}$.

1. **Prefix** dari S mencakup $S[0\dots k]$.
2. **Suffix** dari S mencakup $S[k\dots m - 1]$.

Seperti contoh yang tercantum dalam Slides oleh Rinaldi (2025), misalkan terdapat string sebagai berikut.



Gambar 2.1.1 : String bertuliskan “Andrew”

1. **Prefix** yang memungkinkan untuk S adalah “a”, “an”, “and”, “andr”, “andre”, dan “andrew”.
2. **Suffix** yang memungkinkan untuk S adalah “w”, “ew”, “rew”, “drew”, “ndrew”, andrew”.

2.2 Algoritma *Brute Force* untuk *Pattern Matching*

Sebelum membahas algoritma KMP dan MB dalam penyelesaian *pattern matching*, diperlukan penguasaan mengenai algoritma dasarnya, yaitu **Brute Force** (dapat dikatakan dari algoritma mendasar dari semua penyelesaian masalah). Dalam konteks *pattern matching*, algoritma brute force **memeriksa setiap karakter pada P dan menggeser pattern satu per satu karakter** hingga terjadi **match**. Berikut versi mudahnya.

1. **Posisikan P** sedemikian rupa agar karakter pertama P sejajar dengan karakter pertama T .
2. **Untuk setiap karakter dalam P , cocokkan dengan karakter yang sejajar pada T .** Apabila seluruh karakter sama, kembalikan *true*. Apabila terdapat karakter berbeda, hentikan loop, lanjutkan ke langkah 3.
3. **Geser P sejauh satu karakter ke kanan**, lakukan kembali langkah 2. Ulangi terus selama belum terjadi *match* atau posisi P sudah mencapai $m - \text{panjang}(p) + 1$. Apabila masih belum ditemukan match, kembalikan *false*.

Contohnya, apabila $T = \text{"Mahasiswa ITB."}$ dan $P = \text{"swa"}$, berikut ilustrasinya.

Mahasiswa ITB	
1	swa
2	swa
3	swa
4	swa
5	swa
6	swa
7	swa (Match) , → true

Tabel 2.2.1 Ilustrasi algoritma Brute Force

2.3 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP), merupakan salah satu algoritma *pattern matching* yang mencari *pattern* di dalam *text* dengan urutan dari kiri ke kanan. Bedanya dengan *brute force*, KMP menggeser *pattern* dengan cara yang lebih cerdas. (Informasi mengenai sejarah algoritma ini dapat dibaca langsung pada slide referensi pada daftar pustaka).

Munculnya ide algoritma ini berasal dari pertanyaan “berapa *shift* terjauh yang dapat dilakukan untuk menghindari perbandingan yang *mubazir*?” Ternyata jawabannya adalah prefix **terbesar** (terpanjang) dari $P[0\dots j - 1]$ yang juga merupakan suffix dari $P[1\dots j - 1]$.

Gambar 2.3.1 Ilustrasi shifting yang lebih “cerdas” pada algoritma KMP

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

Jumlah pergeseran = $\text{panjang}(P) - \text{panjang}(\text{Prefix})$, di mana Prefix sesuai spesifikasi di atas.

Pergeseran dengan cara tersebut dapat dilakukan karena **Prefix terpanjang tersebut juga menjadi suffix dari bagian belakang T yang sesuai pattern**, sehingga tidak perlu dicek ulang.

Dalam algoritma KMP, **border function** $b(k)$ didefinisikan sebagai ukuran terbesar prefix $P[0\dots j - 1]$ yang juga merupakan suffix dari $P[1\dots j - 1]$, sering disebut juga sebagai *failure function*.

(k = j-1)						
j	0	1	2	3	4	5
P[j]	a	b	a	a	b	a
k	0	1	2	3	4	
b(k)	0	0	1	1	2	

b(k) is the size of the largest border.

Gambar 2.3.2 Contoh border Function

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf))

1. j : Letak terjadinya mismatch.
2. $k = j - 1$

Berikut merupakan algoritma KMP dalam pseudocode.

```

function KMP(T: String, P: String) → integer
    KAMUS
        ALGORITMA
        b ← COMPUTE_BORDER_FUNCTION(P)
        n ← length(T)
        m ← length(P)
        i ← 0      // index for text T
        j ← 0      // index for pattern P

        while i < n:
            if T[i] == P[j]:
                if j == m - 1:
                    return i - m + 1 // match found
                i ← i + 1
                j ← j + 1
            else if j > 0:
                j ← b[j - 1]
            else:
                i ← i + 1

        return -1 // no match found
    
```

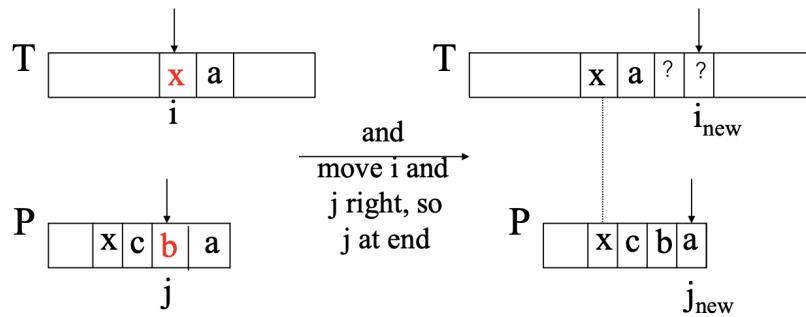
Tabel 2.3.1 Algoritma KMP

Ketika terjadi mismatch selain selain di karakter pertama, pattern akan loncat sesuai border function.

2.4 Algoritma Boyer-Moore

Algoritma Boyer-Moore merupakan algoritma pattern matching yang mencocokkan setiap karakter dalam pattern dalam urutan mundur (dari belakang ke depan). Dalam algoritma ini, dihitung *last occurrence* dari setiap huruf di dalam pattern. Ketika terjadi mismatch, terdapat 3 kemungkinan yang mungkin terjadi, yaitu

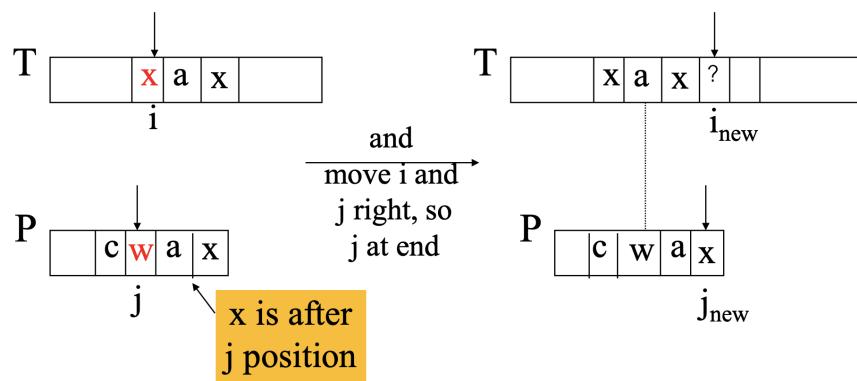
1. Terdapat huruf pada text (di tempat terjadi mismatch) di bagian depan pattern.



Gambar 2.4.1 Terdapat x di bagian kiri terjadinya mismatch

Sumber : Rinaldi Munir

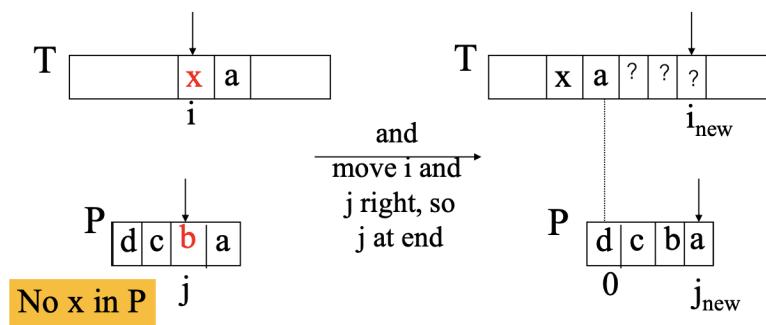
2. Terdapat huruf pada text (di tempat terjadi mismatch) di bagian belakang pattern.



Gambar 2.4.2 Terdapat x di bagian kanan terjadinya mismatch.

Sumber : Rinaldi Munir

3. Pattern tidak mengandung huruf pada text di tempat terjadi mismatch.



Gambar 2.4.3 Tidak terdapat x pada pattern.

Sumber : Rinaldi Munir

Setiap kemungkinan tersebut menimbulkan aksi yang berbeda-beda.

1. Ketika terjadi **kasus 1**, sejajarkan huruf tempat terjadi mismatch pada text dengan *last occurrence* huruf yang sama pada pattern.
2. Ketika terjadi **kasus 2**, geser pattern 1 karakter ke kanan.

3. Ketika terjadi **kasus 3**, geser seluruh pattern sehingga huruf pertama pattern sejajar dengan $i + 1$, di mana i adalah tempat terjadinya mismatch sebelumnya.

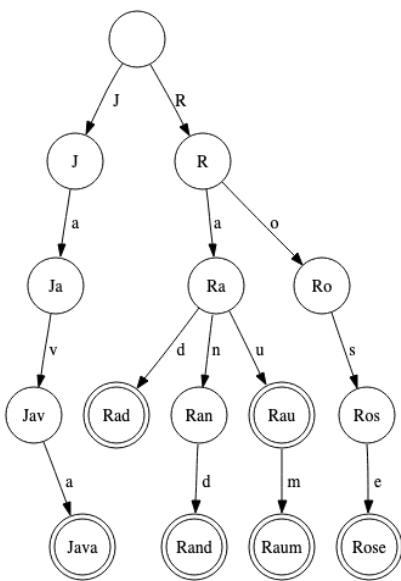
2.5 Algoritma Aho-Corasick

Algoritma **Aho-Corasick** merupakan algoritma string (pattern) matching yang sangat efisien untuk menemukan semua kemunculan dari sekumpulan kata kunci di dalam sebuah teks. Keunggulan utama dari algoritma ini adalah kemampuannya untuk memproses teks dalam **satu kali jalan** (*one single pass*), tidak peduli sebanyak apapun kata kunci yang dicari. Struktur data yang terdapat pada algoritma ini yaitu **trie** (sering juga dikenal sebagai *keyword tree*) yang menggunakan konsep ***finite automaton*** yang dilengkapi dengan **failure links**.

Algoritma ini terdiri dari dua fase utama, yaitu **fase konstruksi** dan **fase pencocokan**. Pada fase Konstruksi, dilakukan pembentukan **trie** dan penambahan **failure links**. Struktur data trie secara umum sebagai berikut.

1. **Root Node** → merepresentasikan **string kosong**.
2. **Path** → setiap jalur dari root ke sebuah node merepresentasikan **prefix**.
3. **Node output** → menandakan akhir suatu kata kunci.

Berikut merupakan contoh struktur data trie yang merepresentasikan enam suku kata, yaitu Java, Rad, Rand, Raum, Rau, dan Rose. Pada trie, setiap huruf mengonstruksi satu kata dengan suatu stopper (bisa apa saja), yang pada gambar ini diilustrasikan sebagai *double circle*. Setiap stopper berarti terdapat satu kata dari prefix tersebut.



Gambar 2.5.1 Contoh gambar Trie yang digunakan untuk algoritma Aho-Corasick

Sumber : <https://upload.wikimedia.org/wikipedia/commons/e/e2/Trie.svg>

Tahap berikutnya adalah **membangun failure links**. *Failure link* dari sebuah node (misalkan u) menunjuk ke node lain dalam trie merepresentasikan *longest proper suffix* dari string yang diwakili oleh node U, yang juga merupakan prefix dalam kamus. Tujuan dibuat failure links adalah jika terjadi

ketidakcocokan, program dapat melanjutkan iterasi mengikuti *failure link* tersebut, inilah mengapa algoritma ini bisa mencari semua kecocokan dalam satu kali jalan.

1. Failure link dari root menunjuk ke dirinya sendiri.
2. Untuk setiap node u , kita melihat failure-link dari parent-nya. Link ini akan menunjukkan suffix terpanjang dari string induk, yang memungkinkan pencarian suffix terpanjang untuk string di node u .

Setelah struktur data berhasil dibuat, fase kedua adalah **pencocokan string**. Berikut kira-kira algoritma pencocokan string untuk Aho-Corasick.

1. Mulai pencarian dari **root** automaton dan karakter pertama pada T (teks).
2. Untuk setiap $c : T$ (c dalam teks), lakukan dua kemungkinan berikut.
 - a. Jika ada anak dari node ini dengan karakter c , **pindah ke node anak tersebut**.
 - b. Jika tidak ada, ikuti **failure link** dari node saat ini dan ulangi pengecekan dengan karakter yang sama (c). Lakukan berulang sampai **menemukan transisi** atau **kembali ke root**.
3. Setelah pindah ke node baru, cek apakah node tersebut merupakan **node output**.
 - a. Jika ya, sebuah kata kunci telah ditemukan. Catat match tersebut.
 - b. Else, ikuti failure link. Jika ada node output di sepanjang rantai tersebut, catat match tersebut. (ini untuk menangani kasus kata yang menjadi suffix kata lain).
4. Ulangi proses sampai **semua karakter selesai dibaca**.

Kompleksitas algoritma Aho-Corasick adalah $O(m + n + k)$, di mana m adalah total panjang dari semua kata kunci dalam kamus, n adalah panjang teks, dan k adalah jumlah kemunculan kata kunci.

BAB III

Analisis Pemecahan Masalah

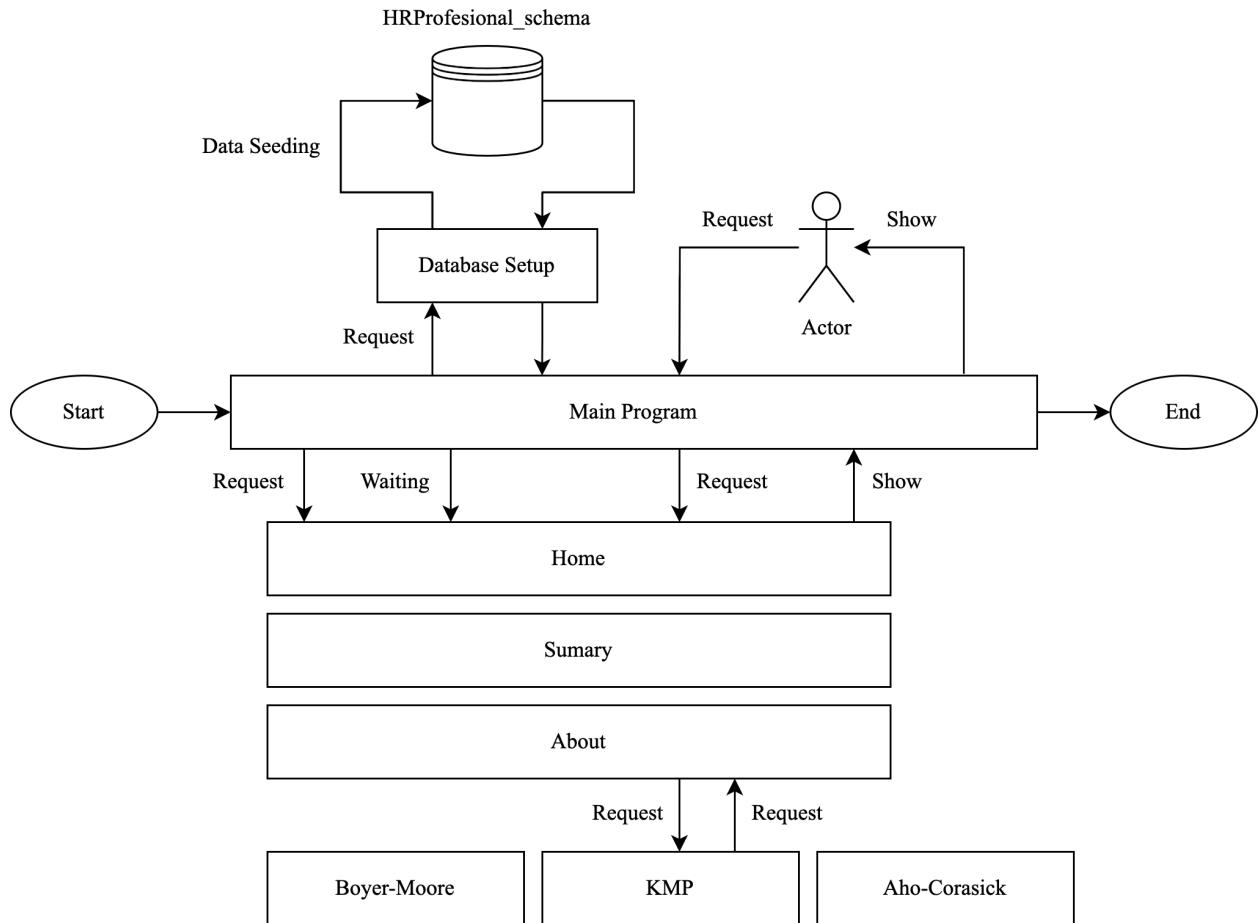
3.1 Dekomposisi Masalah

Dalam pengembangan aplikasi ini, masalah didekomposisi menjadi tiga kelompok utama, yaitu ***data passing***, ***algorithm***, dan ***UI***. Dengan rincian sebagai berikut.

1. **Data passing** mencakup segala hal yang berhubungan dengan pembacaan data dari file .pdf, inisialisasi database, *data seeding*, hingga menghasilkan “objek bahan” yang dapat diterima sebagai parameter oleh algoritma.
2. **Algoritma** mencakup hal-hal yang berhubungan dengan logika pemrograman berdasarkan dasar teori, untuk memproses data input (yang pada proyek ini berupa string atau list of strings), menjadi suatu output yang diinginkan. Selain itu, bagian ini mencakup pengaturan parameter passing untuk menghubungkan *backend* dengan *frontend* (integrasi).
3. **UI** mencakup responsivitas tampilan, kenyamanan penggunaan aplikasi, dan hal-hal lain yang berhubungan erat dengan interaksi langsung antara program dengan pengguna. Perancangan UI mencakup seluruh elemen visual dan animasi interaktif, mulai dari inisialisasi (pemanggilan fungsi) program utama yang menjadi tampilan utama bagi pengguna, pemanggilan fungsi untuk pemrosesan data, hingga menampilkan hasil pemrosesan data tersebut.

3.2 Alur Umum

Berikut merupakan alur pengolahan input berupa file-file .pdf (yang isinya *curriculum vitae*) menjadi summary yang memudahkan pengguna untuk mencari CV sesuai dengan kriteria yang diinginkan. Program akan meminta pengguna untuk memasukkan **kata-kata kunci** yang ingin dicari dengan **tanda koma (’,’)** sebagai pemisah setiap kata kunci, algoritma yang ingin digunakan (KMP, BM, atau Aho-Corasick), dan jumlah file yang ingin ditampilkan. Program akan memproses query tersebut dari data dan memunculkannya dalam bentuk ***CV card*** yang berisi nama pelamar dan jumlah kecocokan kata yang berhasil ditemukan. Pengguna dapat berinteraksi dengan *CV Card* untuk melihat **detail lebih lanjut (CV Summary)** mengenai lamaran kerja tersebut. Berikut merupakan sebuah ***data flow diagram*** sederhana untuk menggambarkan skema berjalannya program.



Gambar 3.2.1 : Data flow diagram keberjalanannya program.
Dibuat dengan draw.io

3.3 Desain Algoritma

Untuk ketiga algoritma yang tersedia (**KMP**, **BM**, dan **Aho-Corasick**), digunakan rancangan implementasi sebagai berikut.

1. Algoritma **KMP** dan **BM**, menerima input berupa string dan mengembalikan list of integer. List of integer merepresentasikan tempat-tempat di mana match ditemukan. Untuk mengakses jumlah *occurrences*, panjang dari list dapat digunakan.
2. Algoritma **Aho-Corasick**, menerima input berupa list of strings dan mengembalikan list of integer, sesuai dengan kemampuannya mengiterasi seluruh kata dalam satu kali run, pengaksesan indeks dan jumlah match sama dengan algoritma lain.

BAB IV

Implementasi dan Pengujian

4.1 Implementasi

4.1.1 Struktur Kode Program

```

Tubes3_HRProfesional/
├── data/                                # Folder untuk menyimpan file dataset
├── doc/                                  # Folder dokumentasi
│   └── [laporan atau file PDF lain] # Contoh: laporan.pdf
└── src/                                   # Source code utama
    ├── backend/
    │   ├── aho_corasick.py      # Algoritma Aho-Corasick
    │   ├── boyer_moore.py       # Algoritma Boyer-Moore
    │   ├── extract_summary.py   # Ekstraksi summary dari data applicant
    │   ├── fetch_from_db.py     # Ambil data profil applicant dari DB
    │   ├── knuth_morris_pratt.py # Algoritma KMP
    │   ├── levenshtein_distance.py # Algoritma Levenshtein
    │   ├── pdf_to_db.py          # Parsing PDF ke database
    │   ├── pdf_to_string.py      # Konversi isi PDF ke string
    │   ├── search_controller.py  # Integrasi pencarian dari frontend
    │   └── search_logic.py       # Logika pencarian/top-k
    ├── data/
    │   ├── setup_data.py          # Script setup database/data awal
    │   └── schema.sql             # Skema database SQL
    └── frontend/
        ├── about.py               # Halaman tentang pembuat aplikasi
        ├── cv.py                  # Halaman lihat CV
        ├── home.py                # Halaman utama
        ├── summary.py              # Halaman ringkasan hasil pencarian
        └── utils.py                # Fungsi utilitas untuk frontend
    └── run_app.py                      # Entry point aplikasi (Flet main app)
└── README.md

```

4.1.2 Implementasi Kode Program

- aho_corasick.py

Fungsi di kode ini membangun struktur “trie” berisi semua keyword yang dicari, lalu menambahkan “failure links” untuk tiap node agar saat karakter tak cocok bisa lompat ke posisi yang tepat tanpa mulai dari akar. Setelah itu, aho_corasick akan berjalan melalui tiap karakter teks, bergerak di dalam trie atau mengikuti tautan kegagalan, dan setiap kali mencapai node dengan daftar output, ia menambah hitungan untuk keyword yang ditemukan. Hasil akhirnya adalah kamus jumlah kemunculan tiap keyword di teks CV.

```
# src/backend/aho_corasick.py
```

```

from collections import deque

def build_trie(keywords: list[str]) -> dict:
    trie = {}
    for keyword in keywords:
        node = trie
        for char in keyword:
            node = node.setdefault(char, {})
        node.setdefault('_output', []).append(keyword)
    return trie

def build_failure_links(trie: dict):
    queue = deque()
    for char, node in trie.items():
        if char != '_output':
            node['_failure'] = trie
            queue.append(node)

    while queue:
        current_node = queue.popleft()
        for char, next_node in current_node.items():
            if char == '_output' or char == '_failure':
                continue

            failure_node = current_node['_failure']
            while char not in failure_node and failure_node is not trie:
                failure_node = failure_node['_failure']

            if char in failure_node:
                next_node['_failure'] = failure_node[char]
            else:
                next_node['_failure'] = trie

            output = next_node['_failure'].get('_output', [])
            next_node.setdefault('_output', []).extend(output)

```

```

queue.append(next_node)

def aho_corasick(text:str, keywords: list[str]) -> dict:
    if not keywords:
        return {}

    trie = build_trie(keywords)
    build_failure_links(trie)

    results = {keyword: 0 for keyword in keywords}
    current_node = trie

    for char in text:
        while char not in current_node and current_node is not trie:
            current_node = current_node['_failure']

        if char in current_node:
            current_node = current_node[char]
        else:
            current_node = trie

        if '_output' in current_node:
            for keyword in current_node['_output']:
                results[keyword] += 1

    return results

```

- boyer_moore.py

Kode ini menjalankan algoritma Boyer-Moore untuk mencari satu pola di dalam teks dengan cepat: ia membangun tabel “last occurrence” untuk mencatat posisi terakhir tiap karakter dalam pola, lalu saat memindai teks ia mulai mencocokkan dari ujung pola dan, jika terjadi *mismatch*, melompat lebih jauh berdasarkan informasi tabel tersebut. Hasilnya adalah daftar indeks di mana pola muncul, tanpa harus memeriksa setiap karakter satu per satu.

```
# src/backend/boyer_moore.py
```

```
from .pdf_to_string import pdf_to_string

def compute_last_occurrence(pattern: str) -> dict:
    last_occurrence = {}
    for i, char in enumerate(pattern):
        last_occurrence[char] = i
    return last_occurrence

def boyer_moore(text: str, pattern: str) -> list[int]:
    n = len(text)
    m = len(pattern)

    if m == 0 or n == 0 or m > n:
        return []

    last_occurrence = compute_last_occurrence(pattern)
    found_indexes = []

    # Start searching from the end of the pattern
    i = m - 1

    while i < n:
        # Pattern index
        j = m - 1

        # Comparison index
        k = i

        while j >= 0 and text[k] == pattern[j]:
            j -= 1
            k -= 1

        if j < 0:
            # Match found
            found_indexes.append(k + 1)
```

```

        # Shift the pattern to the right by the length of the pattern
        i += 1

    else:
        # Mismatch found, Character-Jump
        mismatched_char = text[i]
        temp_last_occurrence = last_occurrence.get(mismatched_char, -1)

        # Calculate the shift based on the last occurrence of the mismatched
        character
        shift = max(1, j - temp_last_occurrence)
        i += shift
    return found_indexes

```

- extract_summary.py

extract_summary.py mengembalikan isi CV yang sudah diubah menjadi string menjadi beberapa section sesuai spesifikasi, yaitu skill, experience, education, dan summary. Nama nama kemungkinan header dicatat dalam sebuah list yang bernama NEXT_HEADER, dan fungsi extract_[nama section] akan mengambil string dari bagian header yang dituju sampai sebelum anggota list NEXT_HEADER.

```

import re
from . import pdf_to_string

NEXT_HEADERS = (
    r'(?P<summary>summary|professional summary|profile|highlights|skills?|technical skills?)|'
    r'work\s+history|work\s+experience|professional\s+experience|experience|'
    r'education|education\s+and\s+training|academics?|accomplishments?|projects?|'
    r'affiliations|interests?)'
)

def grab_section(text, header):
    pattern = rf'^\s*{header}\s*:\s*[\r\n]+(.*)\s*(?=\s*{NEXT_HEADERS}\s*:\s*\$|\Z)'

    #
    rf'^\s*{header}\s*:\s*[\r\n]+(.*)\s*(?=\s*{NEXT_HEADERS}\s*:\s*\b|\Z)'


```

```

m = re.search(pattern, text, re.I | re.M | re.S )
return m.group(1).strip() if m else ''

def extract_skills(text):
    skills_raw = grab_section(text, r'(?s:skills?|technical\s+skills?)')
    skills = [s.strip() for chunk in re.split(r'\n', skills_raw)
              if (s := chunk.strip())]
    return skills

def extract_summary(text):
    summary_raw = grab_section(text,
r'(?s:summary|professional\s+summary|career\s+overview|highlights)')
    summary = [s.strip() for chunk in re.split(r'\n', summary_raw)
              if (s := chunk.strip())]
    return summary

def extract_experience(text):
    exp_raw = grab_section(text, r'(?s:experience|professional\s+experience|work
history|work\s+experience)')
    if not exp_raw:
        return []

    # date range: "Jan 2014 – Present" / "01/2014 - 03/2016" / "2018 - 2020" /
Jan 2024 to Present
    date_pattern = re.compile(
        r'([A-Za-z]{3,9})\s+\d{4}\s*(?:(?:-|to)\s*(?:[A-Za-z]{3,9})\s+\d{4}|present|current)|'
        r'\d{1,2}/\d{4}\s*(?:(?:-|to)\s*(?:\d{1,2}/\d{4}|present|current))|'
        r'\d{4}\s*(?:(?:-|to)\s*(?:\d{4}|present|current)))',
        re.I
    )

    parts = date_pattern.split(exp_raw)
    jobs = []

```

```

for i in range(1, len(parts), 2):
    date_range = parts[i].strip()
    job_chunk = parts[i + 1].strip()

    if not job_chunk:
        continue

    lines = [ln.strip() for ln in job_chunk.splitlines() if ln.strip()]
    company_title = lines[0] if lines else ''

    if '-' in company_title or '-' in company_title or '-' in company_title:
        sep = '-' if '-' in company_title else ('-' if '-' in company_title else '-')
    company, _, after = company_title.partition(sep)
    company = company.strip()
    words = after.strip().split()
    location = ' '.join(words[:2]) if len(words) >= 2 else ''
    job_title = ' '.join(words[2:]) if len(words) > 2 else ''
else:
    company, location, job_title = company_title, '', ''

responsibilities = lines[1:] if len(lines) > 1 else []
jobs.append({
    'date_range': date_range,
    'company': company,
    'location': location,
    'job_title': job_title,
    'responsibilities': responsibilities
})

return jobs

def extract_education(text):
    education_raw = grab_section(text, r'(?P<education>education|education\s+and\s+training)')

```

```
return [ln.strip() for ln in education_raw.splitlines() if ln.strip()]

def parse_resume(text):
    result = {
        'skills': extract_skills(text),
        'summary': extract_summary(text),
        'experience': extract_experience(text),
        'education': extract_education(text)
    }
    return result

def print_parse_result(parsed_data):
    print("==> RESUME PARING RESULT ==>\n\n")

    print("SKILLS:\n")
    for skill in parsed_data['skills']:
        print(f"\t{skill}")
    print("\n")

    print("SUMMARY:\n")
    for summary_line in parsed_data['summary']:
        print(f"\t{summary_line}")
    print("\n")

    print("EXPERIENCE:\n")
    for job in parsed_data['experience']:
        print(f"\t{job['date_range']}")
        print(f"\t{job['company']}")
        if job['location']:
            print(f"\t\t{job['location']}")
        if job['job_title']:
            print(f"\t\t{job['job_title']}")

    if job['responsibilities']:
        print("Responsibilities:\n")
```

```

for resp in job['responsibilities']:
    print(f"    • {resp}")

print("EDUCATION: ")
for education_line in parsed_data['education']:
    print(f"{education_line}")
print("\n")

```

- fetch_from_db.py

fetch_from_db.py adalah memiliki fungsi get_applicant_by_cv untuk mengambil data *profile* dan detail *applicant* dari basis data. get_applicant_by_cv mengambil data berdasarkan parameter path dari file CV karena path dipastikan unik.

```

import mysql.connector, string

def get_applicant_by_cv_path(cv_path: string):
    db = {
        "host":      "localhost",
        "user":      "hr_admin",
        "password":  "",
        "database":  "HRProfesional_schema",
        "charset":   "utf8mb4"
    }
    try:
        connection_configs = [
            {**db, "auth_plugin": "mysql_native_password", "use_pure": True},
            {**db, "use_pure": True},
            db
        ]

        for config in connection_configs:
            try:
                print(f"Trying connection with config: {list(config.keys())}")
                conn = mysql.connector.connect(**config)
                print("Connection successful!")

```

```
        break

    except mysql.connector.Error as e:
        print(f"Connection attempt failed: {e}")
        continue

if not conn:
    print("Gagal koneksi ke database.")
    return

cur = conn.cursor(dictionary=True)
normalize_cv_path = cv_path.replace("\\", "/")
print(f"Mencari pelamar dengan CV path: {normalize_cv_path}")

cur.execute("""
    SELECT ap.applicant_id, ap.first_name, ap.last_name, ap.date_of_birth,
ap.address, ap.phone_number,
            ad.detail_id, ad.application_role, ad.cv_path
    FROM ApplicantProfile ap
    LEFT JOIN ApplicationDetail ad ON ap.applicant_id = ad.applicant_id
    WHERE ad.cv_path = %s
""", (normalize_cv_path,))

result = cur.fetchall()
cur.close()
conn.close()

result_row = []
if result:
    for row in result:
        print("Data Pelamar:")
        print(f"ID: {row['applicant_id']}")
        print(f"Nama: {row['first_name']} {row['last_name']}")
        print(f"Tanggal Lahir: {row['date_of_birth']}")
        print(f"Alamat: {row['address']}")
        print(f"No. HP: {row['phone_number']}")
```

```

        print(f"Role: {row['application_role']}") 
        print(f"CV Path: {row['cv_path']}") 
        result_row.append(row['applicant_id']) 
        result_row.append(row['first_name'] + " " + row['last_name']) 
        result_row.append(row['date_of_birth']) 
        result_row.append(row['address']) 
        result_row.append(row['phone_number']) 
        result_row.append(row['application_role']) 
        result_row.append(row['cv_path']) 
        print(result_row) 
        return result_row 

    else: 
        print(f"Tidak ditemukan pelamar dengan path {cv_path}") 
        return [] 

except mysql.connector.Error as err: 
    print(f"Kesalahan Database: {err}") 
except Exception as e: 
    print(f"Kesalahan lain: {e}")

```

- knuth_morris_prath.py

Kode ini mengimplementasikan algoritma Knuth–Morris–Pratt untuk mencari pola teks secara efisien: pertama menghitung tabel “border” (seberapa panjang prefiks yang juga merupakan sufiks) untuk pola yang dicari, lalu menggunakan tabel itu untuk melewati perbandingan yang tidak perlu saat mencocokkan pola dalam teks. Hasilnya adalah daftar posisi di mana pola tersebut muncul tanpa harus memeriksa setiap karakter berulang kali.

```

# src/backend/knuth_morris_pratt.py

from .pdf_to_string import pdf_to_string
def compute_border_function(pattern: str) -> list[int]:
    # Compute the border function for the Knuth-Morris-Pratt algorithm (size of the
    # largest prefix which is also a suffix)
    # Returns a list[int] (just like PPT)

```

```

m = len(pattern)

# Initialize the border array with zeros
border_array = [0] * m
j = 0 # Length of the previous longest prefix suffix (the value that will be
filled in border_array[i])
i = 1 # Loop index for filling border_array[i]

while i < m:
    if pattern[i] == pattern[j]:
        # Characters match
        j += 1
        border_array[i] = j
        i += 1
    else:
        if j != 0:
            # If there is a mismatch after j matches, use the previous border
            value
            j = border_array[j - 1]
        else:
            # If there is no match, set border_array[i] to 0 and move to the next
            character
            border_array[i] = 0
        i += 1
return border_array

def knuth_morris_pratt(text: str, pattern: str) -> list[int]:
    n = len(text)
    m = len(pattern)

    if m == 0 or n == 0 or m > n:
        return []

    border_array = compute_border_function(pattern)
    found_indexes = []

```

```

i = 0 # Index for text
j = 0 # Index for pattern

while i < n:
    if pattern[j] == text[i]:
        i += 1
        j += 1

    if j == m:
        # Match found
        found_indexes.append(i - j)
        j = border_array[j - 1]

    # Mismatch handling
    elif i < n and pattern[j] != text[i]:
        if j != 0:
            j = border_array[j - 1]
        else:
            # No match, increment i
            i += 1

return found_indexes

```

- levenshtein_distance.py

Kode ini menghitung berapa banyak operasi (menyisipkan, menghapus, atau mengganti karakter) yang diperlukan untuk mengubah satu kata menjadi kata lain, kemudian mengonversi angka tersebut menjadi persentase kemiripan. Hasilnya menunjukkan seberapa mirip dua string secara kuantitatif.

```

# src/backend/levenshtein_distance.py

import numpy as np

def levenshtein_distance(s1 : str, s2 : str) -> int:
    m, n = len(s1), len(s2)

```

```

# Create (m+1) x (n+1) matrix
dp = np.zeros((m + 1, n + 1), dtype=int)

# Initialize the first row and column
for i in range(m + 1):
    dp[i][0] = i
for j in range(n + 1):
    dp[0][j] = j

# Compute the rest mwheehehehehe
for i in range(1, m + 1):
    for j in range(1, n + 1):
        cost = 0 if s1[i - 1] == s2[j - 1] else 1
        dp[i][j] = min(dp[i - 1][j] + 1,          # Deletion
                        dp[i][j - 1] + 1,          # Insertion
                        dp[i - 1][j - 1] + cost) # Substitution

return dp[m][n]

def calculate_similarity(s1 : str, s2 : str) -> float:
    # Just in case (the input should not be Null)
    if not s1 and not s2:
        return 0.0

    distance = levenshtein_distance(s1, s2)
    max_len = max(len(s1), len(s2))
    return (1 - (distance / max_len)) * 100 if max_len > 0 else 0.0

```

- pdf_to_db.py

pdf_to_db.py adalah kode yang digunakan untuk mengekstrak *role* dari applicant dari CV yang terdapat di dataset. *Role* didapatkan dari fungsi regex yang mengambil kata sebelum kata ‘Summary’ yang terdiri dari huruf-huruf kapital. Selain itu, kode ini melakukan seeding nama depan dan nama belakang applicant menggunakan library Faker.

```

import glob, re, mysql.connector
from faker import Faker

```

```
from pdf_to_string import pdf_to_string

def extract_role(text: str) -> str:
    pattern = r"(?m)^(?P<role>[A-Z0-9 &/]+)\s*Summary"
    match = re.search(pattern, text)

    if match is not None:
        role_raw = match.group("role")
        role_clean = role_raw.title()
        return role_clean
    else:
        lines = []
        for line in text.splitlines():
            stripped = line.strip()
            if stripped:
                lines.append(stripped)

        if lines:
            first_line = lines[0]
            return first_line.title()
        else:
            return None

def process_pdf(pdf_path: str, db: dict):
    text = pdf_to_string(pdf_path)
    if not text:
        print(f"[SKIP] kosong: {pdf_path}")
        return

    # generate data dummy
    role = extract_role(text)
    fake = Faker()
    first = fake.first_name()
    last = fake.last_name()
```

```
try:
    # Try multiple connection approaches
    connection_configs = [
        {**db, "auth_plugin": "mysql_native_password", "use_pure": True},
        {**db, "use_pure": True},
        db
    ]

    conn = None
    for config in connection_configs:
        try:
            print(f"Trying connection with config: {list(config.keys())}")
            conn = mysql.connector.connect(**config)
            print("Connection successful!")
            break
        except mysql.connector.Error as e:
            print(f"Connection attempt failed: {e}")
            continue

    if not conn:
        print("All connection attempts failed")
        return

    cur = conn.cursor(prepared=True)

    # insert ke ApplicantProfile
    cur.execute(
        "INSERT INTO ApplicantProfile (first_name, last_name) VALUES (%s, %s)",
        (first, last)
    )
    applicant_id = cur.lastrowid

    cur.execute(
        "INSERT INTO ApplicationDetail (applicant_id, application_role, cv_path)"
    "
```

```

    "VALUES (%s, %s, %s)",
    (applicant_id, role, pdf_path)
)

conn.commit()
cur.close()
conn.close()

print(f"{pdf_path} dengan id={applicant_id} dan role='{role}'")

except mysql.connector.Error as err:
    print(f"Database error: {err}")
    return
except Exception as e:
    print(f"Error processing {pdf_path}: {e}")
    return

```

- pdf_to_string.py

pdf_to_string adalah fungsi yang mengembalikan isi dari pdf sebagai sebuah string panjang menggunakan library pdfplumber.

```

# src/backend/pdf_to_string.py

import os
import pdfplumber
import re

# Opening ONE file at a time and returns its content as ONE string (REGEX version)
def pdf_to_string(pdf_path: str) -> str:

    # Initial checking
    if not os.path.exists(pdf_path):
        print(f"File not found: {pdf_path}")
        return ""

```

```

return_value = []

try:
    with pdfplumber.open(pdf_path) as pdf:
        for page in pdf.pages:
            page_text = page.extract_text()
            if page_text:
                return_value.append(page_text)
    return ''.join(return_value)

except Exception as e:
    print(f"Error processing PDF file {pdf_path}: {e}")
    return ""

# Non-regex version for matching text
def normalize_text(text:str) -> str:
    lowered_text = text.lower()
    no_newlines_text = lowered_text.replace('\n', ' ').replace('\r', ' ')
    normalized_text = re.sub(r'\s+', ' ', no_newlines_text).strip()
    return normalized_text

```

- search_controller.py

search_controller.py ini mengatur cara aplikasi memuat semua CV dari folder, menyimpan isinya ke cache, lalu saat pengguna mencari, ia mencocokkan kata kunci satu per satu ke tiap CV (dengan opsi exact atau fuzzy), menghitung seberapa banyak kata yang cocok dan seberapa relevan, lalu mengembalikan daftar CV teratas beserta waktu pemrosesan. Front-end tinggal memanggil fungsi ini untuk menampilkan hasil pencarian.

```

# src/backend/search_controller.py

import os
import time
import mysql.connector
from .knuth_morris_pratt import knuth_morris_pratt
from .boyer_moore import boyer_moore
from .search_logic import find_fuzzy_matches

```

```

from .aho_corasick import aho_corasick

def get_db_connection():
    try:
        connection = mysql.connector.connect(
            host="localhost",
            user="hr_admin",
            password="",
            database="HRProfesional_schema"
        )
        return connection
    except mysql.connector.Error as e:
        print(f"Database connection error: {e}")
        return None

def get_path_to_name_map():
    path_map = {}
    connection = get_db_connection()
    if not connection: return path_map

    cursor = connection.cursor()
    query = """
SELECT
    p.first_name,
    p.last_name,
    d.cv_path
FROM
    ApplicantProfile p
JOIN
    ApplicationDetail d ON p.applicant_id = d.applicant_id
WHERE
    d.cv_path IS NOT NULL;
"""

    try:
        cursor.execute(query)
    
```

```

        for first_name, last_name, cv_path in cursor.fetchall():
            normalized_path = cv_path.replace('\\', '/').lower()
            path_map[normalized_path] = f'{first_name} {last_name}'
    except mysql.connector.Error as e:
        print(f"Error fetching names from DB: {e}")
    finally:
        cursor.close()
        connection.close()
    return path_map

_cv_data_cache = []

import concurrent.futures

def process_pdf(file_tuple):
    # All imports must be inside for multiprocessing compatibility
    from .pdf_to_string import pdf_to_string, normalize_text
    import os
    file, file_path = file_tuple
    print(f"Processing file: {file}")
    raw_text = pdf_to_string(file_path)
    name_map = get_path_to_name_map()
    try:
        data_index = file_path.lower().rindex('data/')
        relative_path = file_path[data_index:].replace('\\', '/').lower()
    except ValueError:
        relative_path = file_path.replace('\\', '/').lower()
    db_name = name_map.get(relative_path)
    if raw_text:
        return {
            'path': file_path,
            'name': db_name if db_name else os.path.splitext(file)[0],
            'raw_text': raw_text,
            'normalized_text': normalize_text(raw_text)
        }

```

```
return None

def load_cv_data(directory: str):
    global _cv_data_cache
    if _cv_data_cache:
        return
    print(f"Loading CV data from directory: {directory}")

    pdf_files = []
    for root, _, files in os.walk(directory):
        for file in files:
            if file.lower().endswith('.pdf'):
                file_path = os.path.join(root, file)
                pdf_files.append((file, file_path))

    import multiprocessing
    cpu_count = multiprocessing.cpu_count()
    with concurrent.futures.ProcessPoolExecutor(max_workers=cpu_count) as executor:
        results = list(executor.map(process_pdf, pdf_files))
        _cv_data_cache.extend([r for r in results if r is not None])

    print(f"Loaded {len(_cv_data_cache)} CVs.")

def process_cv(cv, clean_keywords, algorithm, fuzzy_threshold):
    import time

    if algorithm == 'KMP':
        search_function = knuth_morris_pratt
    elif algorithm == 'BM':
        search_function = boyer_moore
    elif algorithm == 'AC':
        search_function = None  # handled separately
    else:
        search_function = knuth_morris_pratt
```

```

current_cv_keyword_counts = {}

current_cv_matched_keywords = set()
keywords_to_fuzzy_check = set(clean_keywords)

if algorithm == 'AC':
    from .aho_corasick import aho_corasick
    ac_results = aho_corasick(cv['normalized_text'], list(clean_keywords))
    for keyword, count in ac_results.items():
        if count > 0:
            current_cv_keyword_counts[keyword] = count
            current_cv_matched_keywords.add(keyword)
            if keyword in keywords_to_fuzzy_check:
                keywords_to_fuzzy_check.remove(keyword)

fuzzy_time = 0
if keywords_to_fuzzy_check:
    fuzzy_start = time.perf_counter()
    for keyword in keywords_to_fuzzy_check:
        print(f"Processing fuzzy keyword (AC): '{keyword}' in CV:
'{cv['name']}''")
        fuzzy_matches = find_fuzzy_matches(keyword, cv['normalized_text'],
fuzzy_threshold)
        if fuzzy_matches:
            best_match_word = fuzzy_matches[0]['word']
            fuzzy_count = cv['normalized_text'].count(best_match_word)
            match_key = f"{keyword} → {best_match_word}"
            current_cv_keyword_counts[match_key] = fuzzy_count
            current_cv_matched_keywords.add(keyword)
    fuzzy_time = time.perf_counter() - fuzzy_start
else:
    fuzzy_time = 0
exact_time = 0 # Not timing separately for AC
else:
    exact_start = time.perf_counter()
    for keyword in clean_keywords:

```

```

print(f"Processing keyword: '{keyword}' in CV: '{cv['name']}'")
matches = search_function(cv['normalized_text'], keyword)
if matches:
    current_cv_keyword_counts[keyword] = len(matches)
    current_cv_matched_keywords.add(keyword)
    if keyword in keywords_to_fuzzy_check:
        keywords_to_fuzzy_check.remove(keyword)
exact_time = time.perf_counter() - exact_start
fuzzy_time = 0
if keywords_to_fuzzy_check:
    fuzzy_start = time.perf_counter()
    for keyword in keywords_to_fuzzy_check:
        print(f"Processing fuzzy keyword: '{keyword}' in CV: '{cv['name']}'")
        fuzzy_matches = find_fuzzy_matches(keyword, cv['normalized_text'],
fuzzy_threshold)
        if fuzzy_matches:
            best_match_word = fuzzy_matches[0]['word']
            frequency_of_best_match =
len(search_function(cv['normalized_text'], best_match_word))
            match_key = f"{keyword} → {best_match_word}"
            current_cv_keyword_counts[match_key] = frequency_of_best_match
            current_cv_matched_keywords.add(keyword)
            fuzzy_time = time.perf_counter() - fuzzy_start

    if current_cv_matched_keywords:
        return {
            'name': cv['name'],
            'path': cv['path'],
            'keyword_counts': current_cv_keyword_counts,
            'relevance_score': len(current_cv_matched_keywords),
            'exact_time': exact_time,
            'fuzzy_time': fuzzy_time
        }
else:
    return None

```

```
def _process_cv_wrapper(args):
    cv, clean_keywords, algorithm, fuzzy_threshold = args
    return process_cv(cv, clean_keywords, algorithm, fuzzy_threshold)

def search_cv_data(keywords: list[str], algorithm: str, top_n: int, fuzzy_threshold: float = 80.0) -> dict:
    start_time = time.perf_counter()
    clean_keywords = {k.strip().lower() for k in keywords if k.strip()}

    all_cv_results = []
    total_exact_time = 0
    total_fuzzy_time = 0

    import concurrent.futures
    args_list = [(cv, clean_keywords, algorithm, fuzzy_threshold) for cv in
    _cv_data_cache]

    with concurrent.futures.ProcessPoolExecutor() as executor:
        results = list(executor.map(_process_cv_wrapper, args_list))
        for res in results:
            if res is not None:
                all_cv_results.append(res)
                total_exact_time += res.get('exact_time', 0)
                total_fuzzy_time += res.get('fuzzy_time', 0)

    sorted_results = sorted(all_cv_results, key=lambda x: x['relevance_score'],
    reverse=True)
    print(f"Found {len(sorted_results)} CVs matching the keywords.")

    return {
        'results': sorted_results,
        'scan_count': len(_cv_data_cache),
        'exact_time': total_exact_time * 1000,
        'fuzzy_time': total_fuzzy_time * 1000}
```

```
}
```

- search_logic.py

Search_logic memiliki fungsi find_fuzzy_matches mengambil semua kata unik dari teks CV, lalu membandingkan setiap kata dengan kata kunci pencarian menggunakan skor kemiripan (Levenshtein). Kata-kata yang punya skor di atas ambang batas (default 80%) dikumpulkan, lalu hasilnya diurutkan dari yang paling mirip ke kurang mirip sebelum dikembalikan.

```
# src/backend/search_logic.py

import re
from .levenshtein_distance import calculate_similarity

def find_fuzzy_matches(query: str, cv_text: str, threshold: float = 80.0) ->
list[dict]:
    # Extract unique words from the CV text (Efficiency matters bozo)
    unique_words = set(re.findall(r'\b\w+\b', cv_text.lower()))
    found_matches = []

    for word in unique_words:
        similarity = calculate_similarity(query.lower(), word)

        if similarity >= threshold:
            found_matches.append({
                'word': word,
                'similarity': similarity
            })

    return sorted(found_matches, key=lambda x: x['similarity'])
```

- setup_database.py

setup_database.py adalah kode untuk membuat database di local pengguna jika belum ada skema bernama HRProfesional atau mengisi skema dengan file dump .sql yang disediakan asisten jika sudah ada.

```
# src/data/setup_database.py

import mysql.connector
from mysql.connector import Error as MySQLError
import os

def setup_database():
    # Preventing the "NameError: name 'connection' is not defined" error
    connection = None
    cursor = None

    try:
        # Connect to MySQL server
        connection = mysql.connector.connect (
            host          = "localhost",
            user          = "hr_admin",
            password      = ""
        )
        cursor = connection.cursor()

        # Create database if it doesn't exist
        cursor.execute("CREATE DATABASE IF NOT EXISTS HRProfesional_schema")
        cursor.execute("USE HRProfesional_schema")
        print("Database created.")

        # Use the database
        cursor.execute("USE HRProfesional_schema")

        # Read schema.sql
        sql_directory = os.path.dirname(__file__)
        schema_path = os.path.join(sql_directory, 'schema.sql')

        with open(schema_path, 'r') as file:

            schema_sql = file.read()
```

```

statements = [s.strip() for s in schema_sql.split(';') if s.strip()]
for stmt in statements:
    cursor.execute(stmt)
    print(f"Executed: {stmt.split()[0]}")

for stmt in statements:
    cursor.execute(stmt)
    print(f"Executed: {stmt.split()[0]}")

# Debug print XD
connection.commit()
print("Database setup completed successfully.")

except MySQLError as e:
    print(f"Error connecting to MySQL server: {e}")

finally:
    if cursor is not None:
        cursor.close()
    if connection is not None and connection.is_connected():
        connection.close()

```

- schema.sql

File .sql untuk membuat basis data sesuai dengan spesifikasi tugas besar. Setelah membuat tabel, mengisinya sesuai dengan dump .sql dari asisten.

```

SET NAMES 'utf8mb4' COLLATE 'utf8mb4_unicode_ci';

SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS ApplicationDetail;
DROP TABLE IF EXISTS ApplicantProfile;

SET FOREIGN_KEY_CHECKS = 1;

CREATE TABLE ApplicantProfile (
    applicant_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50),

```

```

last_name VARCHAR(50),
date_of_birth DATE,
address VARCHAR(255),
phone_number VARCHAR(20)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE ApplicationDetail (
    detail_id INT AUTO_INCREMENT PRIMARY KEY,
    applicant_id INT NOT NULL,
    application_role VARCHAR(100),
    cv_path TEXT,
    FOREIGN KEY (applicant_id) REFERENCES ApplicantProfile(applicant_id)
)ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

```

- about.py

about.py mengimplementasikan halaman About Us yang berisikan informasi mengenai pembuat program.

```

import flet as ft

class About:
    def __init__(self, page: ft.Page):
        self.page = page
        self.page.title = "CV Analyzer App by HRProfesional"
        self.page.vertical_alignment = ft.MainAxisAlignment.START
        self.page.horizontal_alignment = ft.CrossAxisAlignment.START
        self.page.bgcolor = '#395B9D'

    def build_ui(self):
        # Header Section
        def on_home_click(e):
            self.page.go("/home")

        self.home_button = ft.ElevatedButton(
            "Home",
            bgcolor="#FAF7F0",
            color="#000000",
            width=100,
            style=ft.ButtonStyle(
                shape=ft.RoundedRectangleBorder(radius=8),
            ),
            on_click=on_home_click,
        )

```

```
self.header_content = ft.Container(
    content=ft.Row(
        [
            ft.Text("CV Analyzer App by HRProfesional", color="#FAF7F0",
size=36, weight=ft.FontWeight.BOLD),
            ft.Row([self.home_button], alignment=ft.MainAxisAlignment.END,
expand=True)
        ],
        vertical_alignment=ft.CrossAxisAlignment.CENTER,
    ),
    width=self.page.window.width,
    bgcolor='#395B9D',
    padding=ft.padding.symmetric(horizontal=20, vertical=15),
    alignment=ft.alignment.center_left
)

panel_content = ft.Container(
    content=ft.Column(
        [
            ft.Container(
                content=ft.Text("About Us", size=28,
weight=ft.FontWeight.BOLD, color="#256988", text_align=ft.TextAlign.CENTER),
                alignment=ft.alignment.center,
                padding=ft.padding.all(5),
            ),
            ft.Divider(height=1, color="#000000"),
            ft.Container(
                content=ft.Text("HRProfesional", size=24,
weight=ft.FontWeight.BOLD, color="#000000", text_align=ft.TextAlign.CENTER),
                alignment=ft.alignment.center,
                padding=ft.padding.only(top=10, bottom=5),
            ),
            ft.Image(
                src="HRProfesional.jpg",
                width=200,
                height=200,
                fit=ft.ImageFit.CONTAIN,
            ),
        ],
    )
)
```

```
        ft.Container(
            content=ft.Text(
                "Undergraduate Informatics Engineering Students \nfrom
Institut Teknologi Bandung \nwho love Algorithm Strategy",
                size=16,
                color="#000000",
                text_align=ft.TextAlign.CENTER,
                width=500,
                # wrap=True,
            ),
            padding=ft.padding.symmetric(horizontal=20, vertical=10),
            alignment=ft.alignment.center,
        ),
    ],
    spacing=10,
    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
),
bgcolor="#DEE2E2",
padding=ft.padding.all(25),
border_radius=ft.border_radius.all(10),
margin=ft.margin.all(10),
expand=4,
alignment=ft.alignment.center
)

return ft.View(
    route="/about",
    controls=[
        ft.Column(
            [
                self.header_content,
                ft.Container(
                    content=panel_content,
                    alignment=ft.alignment.center,
                    expand=True
                )
            ],
            expand=True,
        )
    ]
)
```

```

    ]
)

```

- cv.py

cv.py menampilkan dokumen CV yang dipilih ke layar dengan tombol untuk kembali ke home atau melihat summary, lalu mengubah setiap halaman PDF jadi gambar agar bisa di-zoom in dan out sesuai keinginan pengguna. Jika file CV tidak ditemukan, ia akan menampilkan pesan kesalahan. Dengan begitu, pengguna bisa membaca CV langsung di aplikasi tanpa perlu membuka program lain.

```

import os
import flet as ft
import fitz
import base64
from . import summary

class CV:
    def __init__(self, page: ft.Page, state: dict):
        self.page = page
        self.state = state
        self.page.title = "CV Analyzer App by HRProfesional"
        self.page.vertical_alignment = ft.MainAxisAlignment.START
        self.page.horizontal_alignment = ft.CrossAxisAlignment.START
        self.page.bgcolor = "#395B9D"
        self.display_scale = 1.0 # skala tampilan awal
        self.scale_step = 0.1      # perubahan zoom

    def build_ui(self):
        selected_cv = self.state.get("selected_cv")
        cv_path = selected_cv.get("path") if selected_cv else None

        if not cv_path:
            return ft.View(
                route="/cv",
                bgcolor=self.page.bgcolor,
                controls=[
                    ft.Column(
                        [
                            ft.Text(

```

```
"No CV selected. Please go back and choose a CV  
first.",  
        color="#FAF7F0",  
        size=24,  
        weight=ft.FontWeight.BOLD,  
        ),  
    ],  
    expand=True,  
)  
]  
)  
  
# Header Section  
def on_home_click(e):  
    self.page.go("/home")  
  
self.home_button = ft.ElevatedButton( # home button  
    "Home",  
    bgcolor="#FAF7F0",  
    color="#000000",  
    width=100,  
    style=ft.ButtonStyle(  
        shape=ft.RoundedRectangleBorder(radius=10),  
    ),  
    on_click=on_home_click,  
)  
  
def on_summary_click(cv_data):  
    self.state["selected_cv"] = cv_data  
    self.page.views.append(summary.Summary(self.page,  
self.state).build_ui())  
    self.page.go("/summary")  
  
self.summary_button = ft.ElevatedButton( # summary button  
    "Summary",  
    bgcolor="#FAF7F0",  
    color="#000000",  
    width=100,  
    style=ft.ButtonStyle(  
        shape=ft.RoundedRectangleBorder(radius=10),  
    ),  
    on_click=on_summary_click,  
)
```

```
        shape=ft.RoundedRectangleBorder(radius=10),
    ),
    on_click=lambda e: on_summary_click(selected_cv)
)

self.header_content = ft.Container(
    content=ft.Row(
        [
            ft.Text("CV Analyzer App by HRProfesional", color="#FAF7F0",
size=36, weight=ft.FontWeight.BOLD),
            ft.Row([self.summary_button, self.home_button], spacing=5,
alignment=ft.MainAxisAlignment.END, expand=True)
        ],
        vertical_alignment=ft.CrossAxisAlignment.CENTER,
    ),
    width=self.page.window.width,
    bgcolor='#395B9D',
    padding=ft.padding.symmetric(horizontal=20, vertical=15),
    alignment=ft.alignment.center_left
)

# PDF Viewer Section
pdf_viewer = ft.Column(
    scroll=ft.ScrollMode.AUTO,
    expand=True,
    spacing=0,
    horizontal_alignment=ft.CrossAxisAlignment.CENTER
)

try:
    project_root = os.path.abspath(os.path.join(os.path.dirname(__file__),
'..', '..'))
    source_path = os.path.join(project_root, cv_path)

    if not os.path.exists(source_path):
        raise FileNotFoundError(f"File tidak ditemukan: {source_path}")

    doc = fitz.open(source_path)
    mat = fitz.Matrix(2.0, 2.0)
```

```

for page_doc in doc:
    pix = page_doc.get_pixmap(matrix=mat)
    img_data = pix.tobytes("png")
    image_base64 = base64.b64encode(img_data).decode('utf-8')

    pdf_page_image = ft.Image(
        src_base64=image_base64,
        width=page_doc.rect.width * self.display_scale,
        fit=ft.ImageFit.FILL,
    )

    pdf_viewer.controls.append(pdf_page_image)

doc.close()

except Exception as e:
    error_message = ft.Text(f"Gagal merender PDF: {e}", color="red",
size=16)
    pdf_viewer.controls.append(error_message)

# Kontrol Zoom
zoom_display = ft.Text(f"{int(self.display_scale * 100)}%", weight=ft.FontWeight.BOLD, color="#000000")

def update_image_sizes():
    for image_control in pdf_viewer.controls:
        if isinstance(image_control, ft.Image):
            image_control.width = 800 * self.display_scale
    zoom_display.value = f"{int(self.display_scale * 100)}%"
    self.page.update()

def zoom_in(e):
    self.display_scale += self.scale_step
    if self.display_scale > 3.0: self.display_scale = 3.0
    update_image_sizes()

def zoom_out(e):
    self.display_scale -= self.scale_step

```

```
if self.display_scale < 0.2: self.display_scale = 0.2
update_image_sizes()

zoom_controls = ft.Row(
    [
        ft.IconButton(ft(Icons.ZOOM_OUT, on_click=zoom_out, tooltip="Zoom In"),
                    zoom_display,
                    ft.IconButton(ft(Icons.ZOOM_IN, on_click=zoom_in, tooltip="Zoom Out")),
        ],
        alignment=ft.MainAxisAlignment.CENTER
)

panel_content= ft.Container(
    content=ft.Column(
        [
            ft.Container(zoom_controls, bgcolor="#FFFFFF",
border_radius=ft.border_radius.all(5)),
            pdf_viewer,
        ],
        expand=True,
        spacing=5,
        horizontal_alignment=ft.CrossAxisAlignment.CENTER,
    ),
    expand=True,
    bgcolor="#DEE2E2",
    alignment=ft.alignment.top_center,
    padding=10
)

return ft.View(
    route="/cv",
    bgcolor=self.page.bgcolor,
    controls=[
        ft.Column(
            [
                self.header_content,
                ft.Container(panel_content, expand=True),
            ]
        )
    ]
)
```

```

        ],
        expand=True,
    )
]
)

def main(page: ft.Page):
    # testing
    mock_cv_path = os.path.join('data', 'Agriculture', '10953078.pdf')

    cv = CV(page, {"selected_cv": {"path": mock_cv_path}})
    page.views.append(cv.build_ui())
    page.update()

if __name__ == "__main__":
    project_root = os.path.abspath(os.path.join(os.path.dirname(__file__), '..',
'..'))
    assets_folder_path = os.path.join(project_root, "assets")

    ft.app(target=main)

```

- home.py

home.py adalah halaman utama dari aplikasi ini. Halaman home berisi search bar yang memungkinkan pengguna untuk memasukkan keyword yang ingin dicari, algoritma yang dipakai, dan jumlah maksimal CV yang dicari. Di sebelah kanan, hasil pencarian akan ditampilkan beserta dengan tombol untuk melihat summary dan file CV. Halaman ini juga memuat tombol ke halaman About Us.

```

import flet as ft
import threading
import os
from flet import WebView

from . import utils
from .summary import Summary
from .cv import CV

from src.backend import search_controller

```

```
from src.data.setup_database import setup_database

class Home:
    def __init__(self, page: ft.Page, state: dict):
        self.page = page
        self.state = state
        self.page.title = "CV Analyzer App by HRProfesional"
        self.page.vertical_alignment = ft.MainAxisAlignment.START
        self.page.horizontal_alignment = ft.CrossAxisAlignment.START
        self.page.bgcolor = '#395B9D'
        self.build_ui()
        setup_database()
        search_controller.load_cv_data("data")

        # Global variables for search
        self.keywords = []
        self.algorithm = "BM"

    def build_ui(self):
        # Header Section
        def on_about_us_click(e):
            self.page.go("/about")

        self.about_us_button = ft.ElevatedButton( # about us button
            "About Us",
            bgcolor="#FAF7F0",
            color="#000000",
            width=100,
            style=ft.ButtonStyle(
                shape=ft.RoundedRectangleBorder(radius=8),
            ),
            on_click=on_about_us_click,
        )

        self.header_content = ft.Container( # header
            content=ft.Row(
                [
                    ft.Text("CV Analyzer App by HRProfesional", color="#FAF7F0"),
                    ft.Text("Search", color="#FAF7F0"),
                ],
            ),
        )

    def search_cv(self, keyword: str):
        if keyword == "":
            return self.state["cv_data"]
        else:
            filtered_cv_data = [cv for cv in self.state["cv_data"] if keyword in cv["name"] or keyword in cv["experience"]]
            return filtered_cv_data
```

```
size=36, weight=ft.FontWeight.BOLD),
        ft.Row([self.about_us_button],
alignment=ft.MainAxisAlignment.END, expand=True)
    ],
vertical_alignment=ft.CrossAxisAlignment.CENTER,
),
width=self.page.window.width,
bgcolor='#395B9D',
padding=ft.padding.symmetric(horizontal=20, vertical=15),
alignment=ft.alignment.center_left
)

# Left Panel - Search Controls
def validate_input(e: None):
    if self.num_applicants_input.value and
self.num_applicants_input.value.strip():
        try:
            # Coba konversi ke integer untuk memeriksa apakah itu angka
            int(self.num_applicants_input.value)
            self.num_applicants_input.error_text = None # Hapus error jika
valid
        except ValueError:
            # Jika gagal (bukan angka), tampilkan error dan set tidak valid
            self.num_applicants_input.error_text = "Input must be a number"
            self.page.update()

    self.keywords_input = ft.TextField( # keywords input
        label="Enter keywords separated by comma",
        border_color="#395B9D",
        border_radius=5,
        bgcolor="#FFFFFF",
        text_style=ft.TextStyle(color="#000000"),
    )

    self.algorithm_options = ft.RadioGroup( # algorithm input
        value="KMP", # default
        content=ft.Row(
            [
                ft.Radio(value="KMP", label="KMP",
                
```

```

label_style=ft.TextStyle(size=16, weight=ft.FontWeight.BOLD, color="#000000"),
active_color="#395B9D"),
            ft.Container(width=20),
            ft.Radio(value="BM", label="BM",
label_style=ft.TextStyle(size=16, weight=ft.FontWeight.BOLD, color="#000000"),
active_color="#395B9D"),
            ft.Container(width=20),
            ft.Radio(value="AC", label="Aho-Corasick",
label_style=ft.TextStyle(size=16, weight=ft.FontWeight.BOLD, color="#000000"),
active_color="#395B9D")
        ],
        alignment=ft.MainAxisAlignment.START,
    )
)

self.num_applicants_input = ft.TextField( # number of applicants input
    label="Enter amount",
    border_color="#395B9D",
    border_radius=5,
    bgcolor="#FFFFFF",
    text_style=ft.TextStyle(color="#000000"),
    on_change=validate_input,
)

```

```

def on_summary_click(cv_data):
    self.state["selected_cv"] = cv_data
    # self.page.views.append(Summary(self.page, self.state).build_ui())
    self.page.go("/summary")

```

```

def on_view_cv_click(cv_data):
    self.state["selected_cv"] = cv_data
    # self.page.views.append(CV(self.page,
self.state["selected_cv"]).build_ui())

```

```

    self.page.go("/cv")

```

```

def update_ui_from_state():
    results_info_text.value = self.state["last_info_text"]
    cv_results_grid.controls.clear()

```

```
if not self.state["search_results"]:
    print("No search results found.")
    cv_results_grid.controls.append(ft.Text("No matching CVs found.",
text_align=ft.TextAlign.CENTER))
else:
    sorted_cv_data = sorted(
        self.state["search_results"],
        key=lambda cv: sum(cv["keyword_counts"].values()),
        reverse=True
    )
    for cv_data in sorted_cv_data:
        summary_handler = lambda _, cv=cv_data: on_summary_click(cv)
        view_cv_handler = lambda _, cv=cv_data: on_view_cv_click(cv)

        cv_results_grid.controls.append(
            utils.create_cv_card(self.page,
                cv_data["name"],
                cv_data["keyword_counts"],
                on_summary_click=summary_handler,
                on_view_cv_click=view_cv_handler,
            )
        )
    self.page.update()

self.search_output = None
def on_search_click(e):
    # 1. Ambil input dari UI
    keywords_str = self.keywords_input.value
    if not keywords_str:
        return # Jangan lakukan apa-apa jika keyword kosong

    keywords = [k.strip() for k in keywords_str.split(',')]
    algorithm = self.algorithm_options.value
    try:
        top_n = int(self.num_applicants_input.value)
    except (ValueError, TypeError):
        top_n = 10 # Default 10 jika input kosong atau tidak valid
```

```

# 2. Panggil controller backend
    self.search_output = search_controller.search_cv_data(keywords,
algorithm, top_n, fuzzy_threshold=80.0)
    self.state["search_results"] = self.search_output['results'][:top_n]
    self.state["last_info_text"] = f"{self.search_output['scan_count']} CVs
scanned in {self.search_output['exact_time'] + self.search_output['fuzzy_time']:.2f}
ms"

# 3. Update UI dengan hasil pencarian
update_ui_from_state()

self.search_button = ft.ElevatedButton( # search button
    "Search",
    bgcolor="#FDF6EC",
    color="#395B9D",
    width=450,
    height=40,
    style=ft.ButtonStyle(
        shape=ft.RoundedRectangleBorder(radius=8),
    ),
    expand=True,
    on_click=on_search_click,
)

# Left Panel - Search Controls
left_panel_content = ft.Container(
    content=ft.Column(
        [
            ft.Text(
                "Finding your desired employee\nis just one-click away!",
                size=24,
                weight=ft.FontWeight.BOLD,
                color="#256988"
            ),
            ft.Container(height=10),
            ft.Text("What are you looking for?", size=16,
weight=ft.FontWeight.W_500, color="#000000"),
            self.keywords_input,
        ]
    )
)

```

```
        ft.Container(height=10),
        ft.Text("Choose the searching algorithm", size=16,
weight=ft.FontWeight.W_500, color="#000000"),
        self.algorithm_options,
        ft.Container(height=10),
        ft.Text("How many applicants do you want?", size=16,
weight=ft.FontWeight.W_500, color="#000000"),
        self.num_applicants_input,
        ft.Container(height=15),
        self.search_button,
    ],
    spacing=10,
),
bgcolor="#DEE2E2",
padding=ft.padding.all(25),
border_radius=ft.border_radius.all(10),
margin=ft.margin.all(10),
)

# Right Panel - Results
results_info_text = ft.Text()

cv_results_grid = ft.GridView(
    runs_count=3,
    max_extent=280,
    child_aspect_ratio=0.85, # Sesuaikan rasio aspek kartu
    spacing=10,
    run_spacing=10,
    padding=10,
    # expand=True
)

update_ui_from_state()

right_panel_content = ft.Container(
    content=ft.Column(
        [
            ft.Container(
                content=ft.Text("Results", size=28,
```

```
weight=ft.FontWeight.BOLD, color="#256988", text_align=ft.TextAlign.CENTER),
            alignment=ft.alignment.center,
            # padding=ft.padding.only(bottom=5),
        ),
        ft.Container(
            content=results_info_text,
            alignment=ft.alignment.center,
            padding=ft.padding.only(bottom=5),
        ),
        ft.Divider(height=1, color="#000000"),
        ft.Container(cv_results_grid, expand=True)
    ],
    spacing=5,
    # expand=True
),
bgcolor="#DEE2E2",
padding=ft.padding.all(25),
border_radius=ft.border_radius.all(10),
margin=ft.margin.all(10),
)

# Main Layout
main_layout = ft.Row(
[
    ft.Container(left_panel_content, expand=2, padding=5), # Panel kiri
mengambil 2 bagian
    ft.Container(right_panel_content, expand=3, padding=5), # Panel
kanan mengambil 3 bagian
],
vertical_alignment=ft.CrossAxisAlignment.START,
# expand=True
)

# self.page.clean()
# self.page.add(
#     ft.Column(
#         [
#             self.header_content,
#             ft.Container(main_layout, expand=True)
#         ]
#     )
# )
```

```

        # ],
        # expand=True
        # )
    # )
# self.page.update()

return ft.View(
    route="/home",
    controls=[
        ft.Column(
            [
                self.header_content,
                ft.Container(main_layout, expand=True),
            ],
            expand=True,
        ),
    ],
)

```

```

def main(page: ft.Page):
    app = Home(page)

if __name__ == "__main__":
    ft.app(target=main)

```

- summary.py

summary.py menampilkan hasil summary (profil, skill, summary, experience, dan education) yang sudah diekstrak dari file CV dari *applicant* yang dipilih. Halaman ini juga menunjukkan CV lain hasil pencarian di sebelah kiri.

```

import flet as ft
from . import utils, cv
from ..backend.extract_summary import parse_resume, print_parse_result
from ..backend.pdf_to_string import pdf_to_string
from ..backend.fetch_from_db import get_applicant_by_cv_path

class Summary:

```

```
def __init__(self, page: ft.Page, state: dict):
    self.page = page
    self.state = state
    self.page.title = "CV Analyzer App by HRProfesional"
    self.page.vertical_alignment = ft.MainAxisAlignment.START
    self.page.horizontal_alignment = ft.CrossAxisAlignment.START
    self.page bgcolor = '#395B9D'

def build_ui(self):
    selected_cv = self.state.get("selected_cv")
    if not selected_cv:
        return ft.View(
            route="/summary",
            bgcolor=self.page bgcolor,
            controls=[
                ft.Text("No CV selected. Please select a CV from the home
page.", color="#FAF7F0", size=24, weight=ft.FontWeight.BOLD)
            ]
        )
    # Header Section
    def on_home_click(e):
        self.page.go("/home")

        self.home_button = ft.ElevatedButton( # home button
            "Home",
            bgcolor="#FAF7F0",
            color="#000000",
            width=100,
            style=ft.ButtonStyle(
                shape=ft.RoundedRectangleBorder(radius=10),
            ),
            on_click=on_home_click,
        )

        self.header_content = ft.Container(
            content=ft.Row(
                [
                    ft.Text("CV Analyzer App by HRProfesional", color="#FAF7F0",
size=36, weight=ft.FontWeight.BOLD),
                ]
            )
        )
```

```

        ft.Row([self.home_button], alignment=ft.MainAxisAlignment.END,
expand=True)
    ],
    vertical_alignment=ft.CrossAxisAlignment.CENTER,
),
width=self.page.window.width,
bgcolor='#395B9D',
padding=ft.padding.symmetric(horizontal=20, vertical=15),
alignment=ft.alignment.center_left
)

# Left Panel - Other CVs
cv_results_grid = ft.Column(
    spacing=10,
    scroll=ft.ScrollMode.AUTO,
    horizontal_alignment=ft.CrossAxisAlignment.START,
    # padding=10,
    expand=True
)

def on_summary_click(cv_data):
    self.state["selected_cv"] = cv_data
    self.page.views.append(Summary(self.page, self.state).build_ui())
    self.page.go("/summary")

def on_view_cv_click(cv_data):
    self.state["selected_cv"] = cv_data
    self.page.views.append(cv.CV(self.page,
self.state["selected_cv"]).build_ui())

    self.page.go("/cv")

for cv_data in self.state.get("search_results", []):
    if cv_data.get("path") != selected_cv.get("path"):
        summary_handler = lambda _, cv=cv_data: on_summary_click(cv)
        view_cv_handler = lambda _, cv=cv_data: on_view_cv_click(cv)

        cv_results_grid.controls.append(
            utils.create_cv_card(self.page,

```

```
        cv_data[ "name" ],
        cv_data[ "keyword_counts" ],
        on_summary_click=summary_handler,
        on_view_cv_click=on_view_cv_click
    )
)

left_panel_content = ft.Container(
    content=ft.Column(
        [
            ft.Text("Other CVs", size=20, weight=ft.FontWeight.BOLD,
color="#DEE2E2", text_align=ft.TextAlign.CENTER),
            # ft.Divider(height=1, color="#000000"),
            cv_results_grid,
        ],
        spacing=5,
        expand=True
    ),
    bgcolor="#395B9D",
    # padding=ft.padding.all(25),
    # border_radius=ft.border_radius.all(10),
    # margin=ft.margin.all(10),
)
)

# Right Panel - Summary
def create_cv_summary_card(content, text):
    # content: SUMMARY, SKILLS, EXPERIENCE, EDUCATION
    return ft.Card(
        height=200,
        content=ft.Container(
            content=ft.Column(
                [
                    ft.Text(content, size=20, weight=ft.FontWeight.BOLD,
color="#000000", text_align=ft.TextAlign.CENTER),
                    ft.Divider(height=1, color="#000000"),
                    ft.Text(text, size=16, color="#000000",
text_align=ft.TextAlign.LEFT),
                ],
                spacing=5,
```

```

        horizontal_alignment=ft.CrossAxisAlignment.START,
        scroll=ft.ScrollMode.AUTO,
        expand=True,
    ),
    padding=ft.padding.all(20),
    alignment=ft.alignment.top_left,
    bgcolor="#FFFFFF",
    border_radius=8,
),
)

cv_summary_grid = ft.Column(
    spacing=30,
    scroll=ft.ScrollMode.AUTO,
    horizontal_alignment=ft.CrossAxisAlignment.START,
    # padding=10
)

cv_path = selected_cv.get("path")
if not cv_path:
    return ft.View(route="/summary", controls=[ft.Text("No search results
yet!")])
text = pdf_to_string(cv_path) if cv_path else ""
parsed_text = parse_resume(text) if text else {}
profile = get_applicant_by_cv_path(cv_path) if cv_path else {}

profile_card = ft.Card(
    height=200,
    content=ft.Container(
        content=ft.Column(
            [
                ft.Text("PROFILE", size=20, weight=ft.FontWeight.BOLD,
color="#000000", text_align=ft.TextAlign.CENTER),
                ft.Divider(height=1, color="#000000"),
                ft.Text(f"Name: {profile[1]} \nDate of Birth: {profile[2]} \nAddress: {profile[3]} \nPhone Number: {profile[4]} \nRole: {profile[5]} \n",
size=16, color="#000000",
text_align=ft.TextAlign.LEFT),
            ],
        )
    )
)

```

```

        spacing=5,
        horizontal_alignment=ft.CrossAxisAlignment.START,
        scroll=ft.ScrollMode.AUTO,
        expand=True,
    ),
    padding=ft.padding.all(20),
    alignment=ft.alignment.top_left,
    bgcolor="#FFFFFF",
    border_radius=8,
),
)

cv_summary_grid.controls.append(profile_card) # profil

summary_string = "\n".join(parsed_text.get('summary', [])) or "Not available"
skills_string = "\n".join(parsed_text.get('skills', [])) or "Not available"
education_string = "\n".join(parsed_text.get('education', [])) or "Not available"
experience_items = []
for job in parsed_text.get('experience', []):
    date_range = job.get('date_range') or '-'
    company = job.get('company') or '-'
    location = job.get('location') or '-'
    job_title = job.get('job_title') or '-'
    responsibilities_list = job.get('responsibilities', [])
    if responsibilities_list:
        responsibilities_text = '\n'.join([f" {resp}" for resp in responsibilities_list])
    else:
        responsibilities_text = "-"
    job_text = (
        f"Date range: {date_range}\n"
        f"Company: {company}\n"
        f"Location: {location}\n"
        f"Job Title: {job_title}\n"
        f"Responsibilities:\n{responsibilities_text}"
    )
    experience_items.append(job_text)
)

```

```
experience_string = "\n\n".join(experience_items) or "Not available"

cv_summary_grid.controls.append(create_cv_summary_card("SUMMARY",
summary_string))
cv_summary_grid.controls.append(create_cv_summary_card("SKILLS",
skills_string))
cv_summary_grid.controls.append(create_cv_summary_card("EXPERIENCE",
experience_string))
cv_summary_grid.controls.append(create_cv_summary_card("EDUCATION",
education_string))

right_panel_content = ft.Container(
    content=ft.Column(
        [
            ft.Row(
                [
                    ft.Container(
                        content=ft.Text("CV Summary", size=28,
weight=ft.FontWeight.BOLD, color="#256988", text_align=ft.TextAlign.CENTER),
                        alignment=ft.alignment.top_left,
                        expand=True,
                    ),
                    ft.Container(
                        content=ft.ElevatedButton("View CV",
style=ft.ButtonStyle(
                                bgcolor="#395B9D",
                                color="#DEE2E2",
                                shadow_color="#395B9D",
shape=ft.RoundedRectangleBorder(radius=8)
),
on_click=lambda _ :
on_view_cv_click(selected_cv)
),
alignment=ft.alignment.top_right,
),
],
vertical_alignment=ft.CrossAxisAlignment.CENTER,
),

```

```
        ft.Divider(height=1, color="#000000"),
        ft.Container(cv_summary_grid, expand=True)
    ],
    spacing=5,
    # expand=True
),
bgcolor="#DEE2E2",
padding=ft.padding.all(25),
border_radius=ft.border_radius.all(10),
margin=ft.margin.all(10),
)

# Main Layout
main_layout = ft.Row(
[
    ft.Container(left_panel_content, expand=1, padding=5),
    ft.Container(right_panel_content, expand=5, padding=5),
],
vertical_alignment=ft.CrossAxisAlignment.START,
expand=True
)
return ft.View(
    route="/summary",
    bgcolor=self.page.bgcolor,
    controls=[
        ft.Column(
            [
                self.header_content,
                ft.Container(main_layout, expand=True),
            ],
            expand=True,
        )
    ]
)

def main(page: ft.Page):
    summary = Summary(page)
    page.views.append(summary.build_ui())
    # page.bgcolor = '#395B9D'
```

```
page.update()

if __name__ == "__main__":
    ft.app(target=main)
```

- utils.py

utils.py menyimpan fungsi pembantu untuk tampilan *frontend*, seperti `create_cv_card` untuk membuat card untuk menampilkan hasil search di `home.py`.

```
import flet as ft

def create_cv_card(page:ft.Page, name, results, on_summary_click=None,
on_view_cv_click=None, height=250):
    def details_dialog(e):
        full_keyword_texts = []
        for i, (keyword, count) in enumerate(results.items()):
            if (count == 1):
                full_keyword_texts.append(
                    ft.Text(f"{i+1}. {keyword}: {count} occurence", size=12,
color="#000000")
                )
            elif (count > 1):
                full_keyword_texts.append(
                    ft.Text(f"{i+1}. {keyword}: {count} occurrences", size=12,
color="#000000")
                )
        dialog_content = ft.Column(
            controls=full_keyword_texts,
            tight=True,
            scroll=ft.ScrollMode.AUTO,
            height=200,
        )
        details_dialog = ft.AlertDialog(
            modal=True,
            title=ft.Text("Matched keywords: "),
            content=dialog_content,
            scrollable=True,
```

```

        actions=[

            ft.TextButton("Close", on_click=lambda e: close_dialog(e),
style=ft.ButtonStyle(
                color="#395B9D",
                text_style=ft.TextStyle(size=14, weight=ft.FontWeight.W_500)
            )
        ],
        actions_alignment=ft.MainAxisAlignment.END,
    )

    def close_dialog(e):
        details_dialog.open = False
        page.update()

        page.dialog = details_dialog
        details_dialog.open = True
        page.update()

    # result adalah dictionary dengan keyword sebagai key dan jumlah kemunculan
    # sebagai value
    total_matches = sum(count for count in results.values() if isinstance(count,
int))
    matched_keywords_text = ft.Container(
        content=ft.Text(f"Matched keywords:", size=12, weight=ft.FontWeight.W_500,
color="#000000"),
        on_click=details_dialog,
        tooltip="See all matched keywords",
    )
    keyword_texts = []
    for i, (keyword, count) in enumerate(results.items()):
        if (i == 3):
            keyword_texts.append(ft.Text(f"... and {len(results) - 3} more
keywords", size=12, color="#000000"))
            break
        if isinstance(count, int) and count == 1:
            keyword_texts.append(
                ft.Text(f"{i+1}. {keyword}: {count} occurence", size=12,
color="#000000")
            )

```

```
)  
    elif isinstance(count, int) and count > 1:  
        keyword_texts.append(  
            ft.Text(f"{i+1}. {keyword}: {count} occurrences", size=12,  
color="#000000")  
        )  
    else:  
        # For fuzzy matches (string), show the string as is  
        keyword_texts.append(  
            ft.Text(f"{i+1}. {keyword}: {count}", size=12, color="#000000")  
        )  
  
    return ft.Card(  
        height = height,  
        content=ft.Container(  
            content=ft.Column(  
                [  
                    ft.Row(  
                        [  
                            ft.Container(  
                                content=ft.Text(name, weight=ft.FontWeight.BOLD,  
size=18, color="#000000"), # nama  
                                alignment=ft.alignment.top_left,  
                                expand=True  
                            ),  
                            ft.Container(  
                                content=ft.Text(f"{total_matches} matches", size=12,  
color="#000000"), # total matches  
                                alignment=ft.alignment.top_right,  
                                expand=False  
                            ),  
                        ],  
                        vertical_alignment=ft.CrossAxisAlignment.START,  
                        spacing=5  
                    ),  
                    ft.Container(height=5),  
                    matched_keywords_text,  
                    ft.Column(keyword_texts, spacing=2, expand=True),  
                    ft.Row(  
                ]  
            )  
        )  
    )
```

```
[  
    ft.Container(  
        content=ft.ElevatedButton("Summary",  
            style=ft.ButtonStyle(  
                bgcolor="#395B9D",  
                color="#DEE2E2",  
                shadow_color="#395B9D",  
  
                shape=ft.RoundedRectangleBorder(radius=8)  
                    ),  
                    on_click=on_summary_click  
                ),  
                alignment=ft.alignment.bottom_left,  
                expand=True,  
            ),  
            ft.Container(  
                content=ft.ElevatedButton("View CV",  
                    style=ft.ButtonStyle(  
                        bgcolor="#395B9D",  
                        color="#DEE2E2",  
                        shadow_color="#395B9D",  
  
                        shape=ft.RoundedRectangleBorder(radius=8)  
                            ),  
                            on_click=on_view_cv_click  
                        ),  
                        alignment=ft.alignment.bottom_right,  
                    ),  
  
    ],  
    vertical_alignment=ft.CrossAxisAlignment.END,  
    spacing=5  
)  
],  
spacing=3  
),  
width=200,  
padding=15,  
bgcolor="#FFFFFF",
```

```
        border_radius=8,  
        shadow=ft.BoxShadow(blur_radius=5, color="#395B9D"))  
    )  
}
```

- run_app.py

File ini berperan sebagai entry point aplikasi yang menginisialisasi page dan state global, menampilkan loading screen sementara, dan membuat route-to-view mapping untuk menghubungkan URL ("/home", "/about", "/summary", "/cv") dengan komponen UI masing-masing.

```
import flet as ft
import sys
import os

sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))

from src.frontend.home import Home
from src.frontend.about import About
from src.frontend.summary import Summary
from src.frontend.cv import CV

app_state = {
    "search_results": [],
    "last_info_text": "",
    "selected_cv": None,
}

def main(page: ft.Page):
    page.title = "CV Analyzer App by HRProfesional"
    page.vertical_alignment = ft.MainAxisAlignment.START
    page.horizontal_alignment = ft.CrossAxisAlignment.START
    page.bgcolor = '#395B9D'

    loading_sign = ft.Container(
        content=ft.Column(
            [
                ft.Text("Loading...", color="#FAF7F0", size=36)
            ]
        )
    )
```

```
weight=ft.FontWeight.BOLD),
            ft.Container(height=10),
            ft.ProgressRing(width=30, height=30, stroke_width=3,
color="#FAF7F0")
        ],
        alignment=ft.MainAxisAlignment.CENTER,
        horizontal_alignment=ft.CrossAxisAlignment.CENTER
    ),
    expand=True
)
page.add(loading_sign)

pages = {
    "/home": Home(page, app_state),
    "/about": About(page),
    "/summary": Summary(page, app_state),
    "/cv": CV(page, app_state)
}

def route_change(route):
    page.views.clear()
    current_view = pages.get(page.route, pages["/home"]).build_ui()
    current_view.bgcolor = page.bgcolor
    page.views.append(current_view)
    page.update()

def view_pop(view):
    page.views.pop()
    top_view = page.views[-1]
    page.go(top_view.route)

page.on_route_change = route_change
page.on_view_pop = view_pop

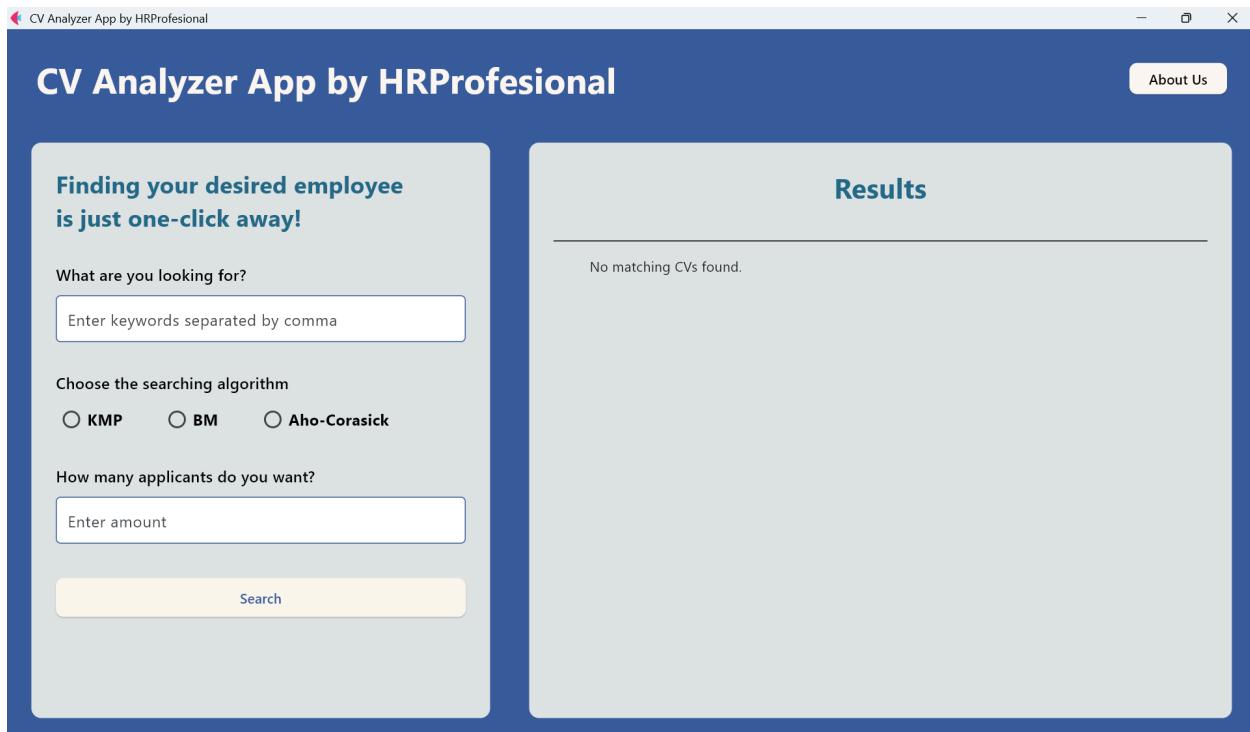
page.go(page.route or "/home")

if __name__ == "__main__":
    assets_folder_path = "assets"
```

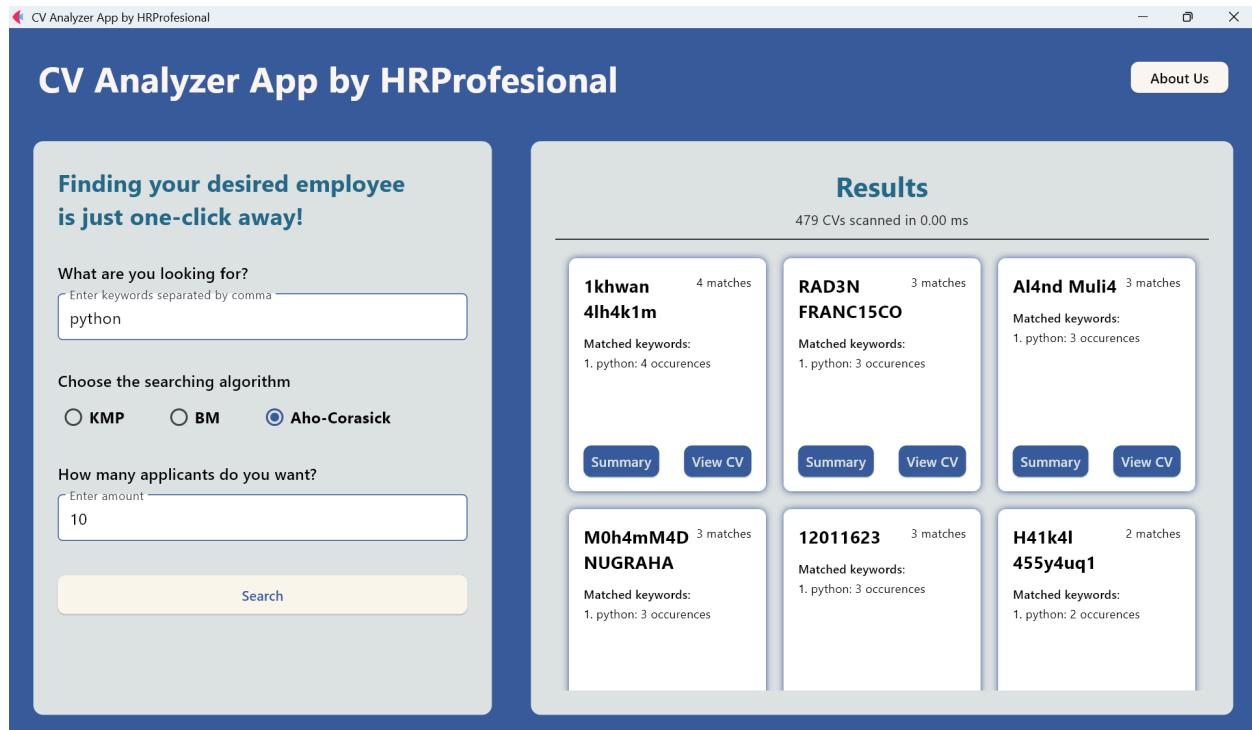
```
ft.app(target=main, assets_dir=assets_folder_path, )
```

4.2 Cara Penggunaan Program

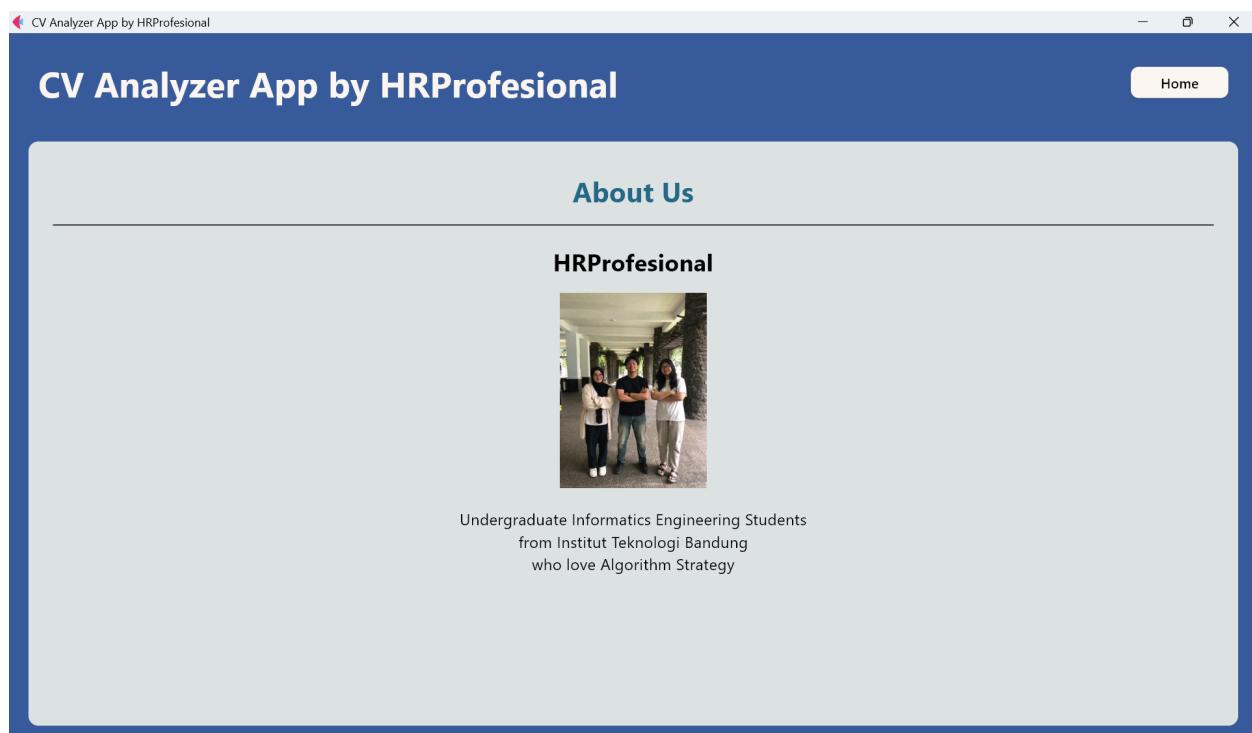
Saat pertama masuk ke dalam aplikasi, pengguna akan diarahkan ke halaman home. Di halaman ini, pengguna dapat memasukkan keyword yang ingin dicari, algoritma yang dipakai, dan jumlah maksimal CV yang dicari. Di sebelah kanan, hasil pencarian akan ditampilkan beserta dengan tombol untuk melihat summary dan file CV. Dari halaman ini juga, dapat diakses halaman About Us serta summary dan file CV dari hasil pencarian.



Gambar 4.1 Tampilan awal halaman Home

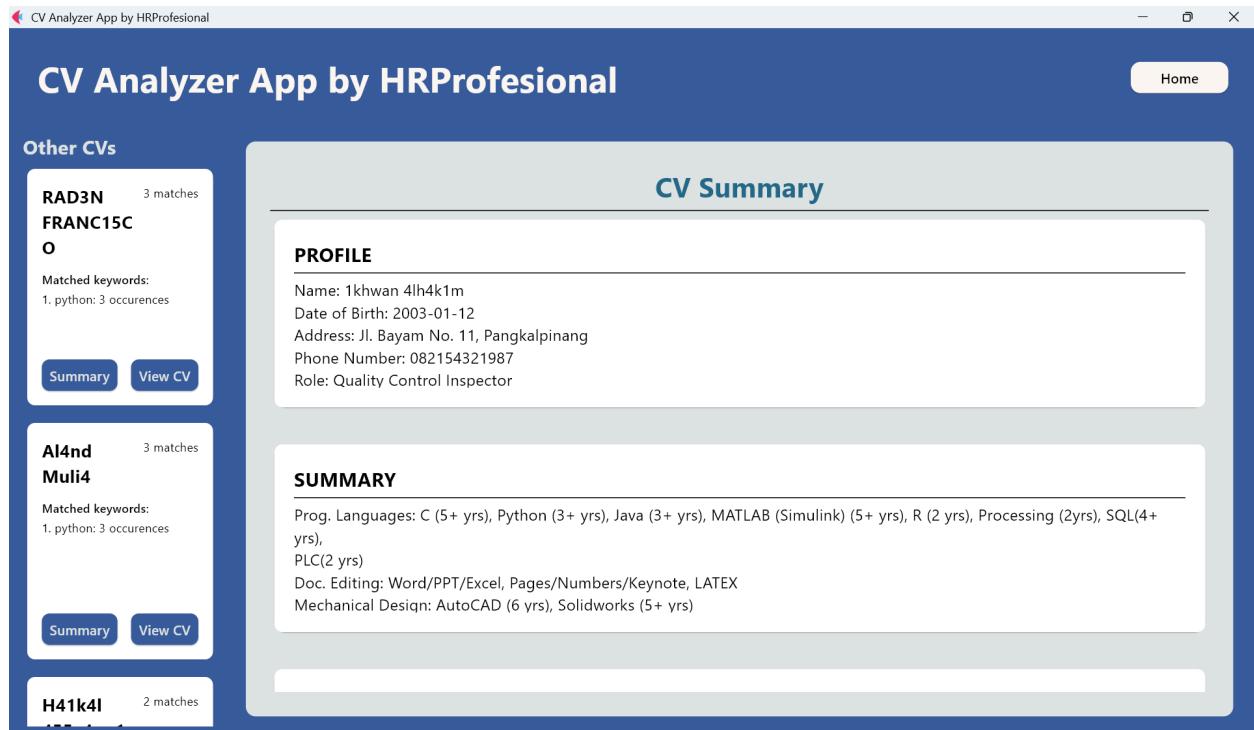


Gambar 4.2 Tampilan halaman Home setelah melakukan pencarian



Gambar 4.3 Tampilan halaman About Us

Ketika pengguna pergi ke page summary untuk sebuah CV, di sebelah kiri akan terlihat CV-CV lain yang juga merupakan hasil dari pencarian. Dari bagian ini, summary serta file CV lainnya juga bisa dilihat.

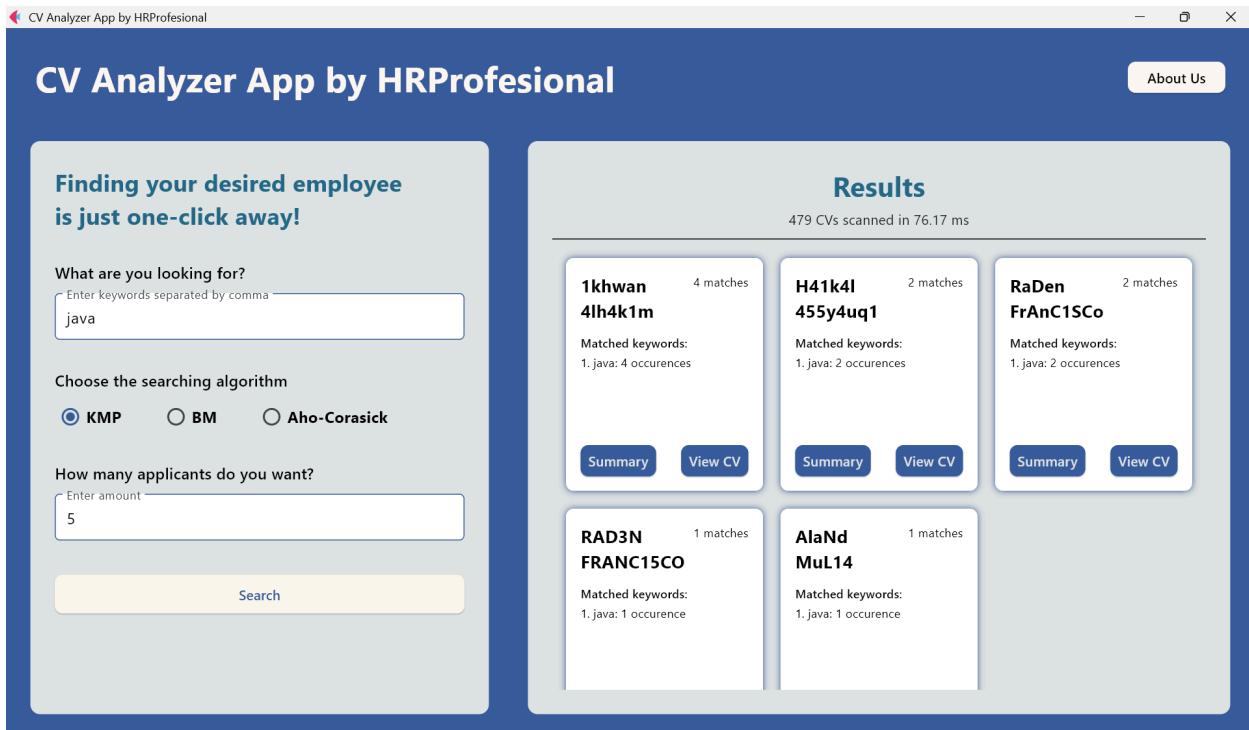


Gambar 4.4 Tampilan halaman Summary

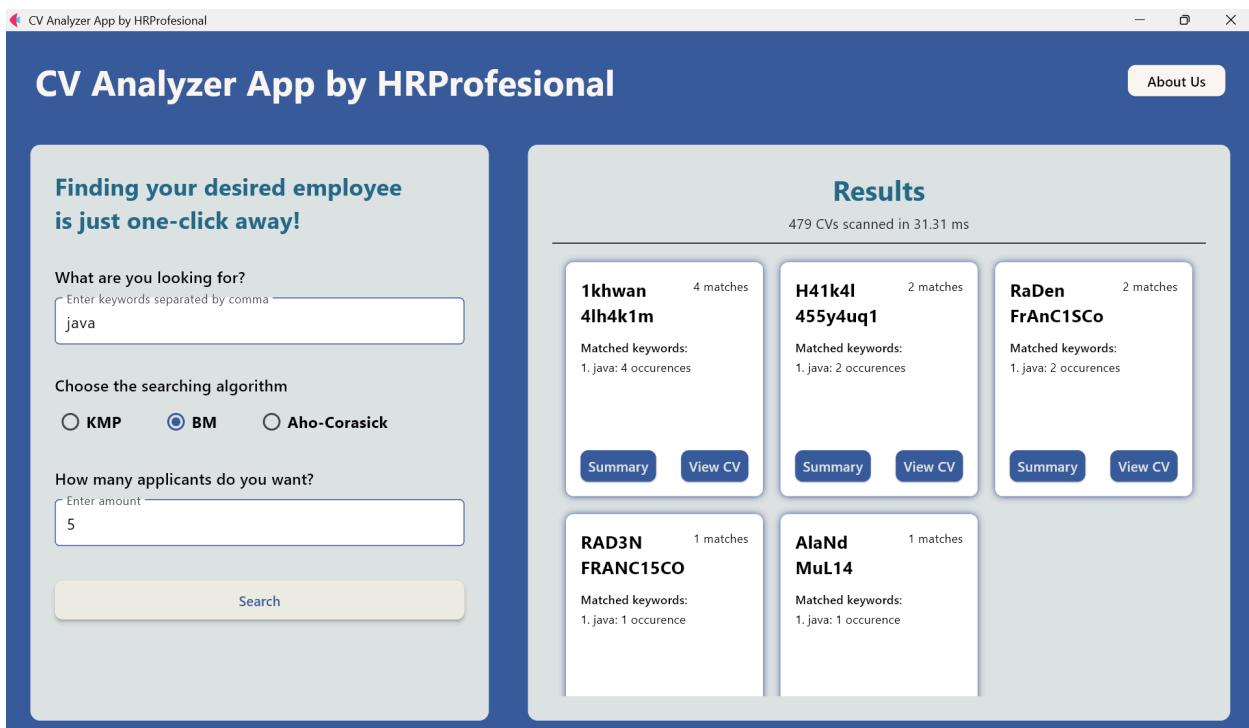
4.3 Pengujian

4.3.1 Pencarian 1 keyword

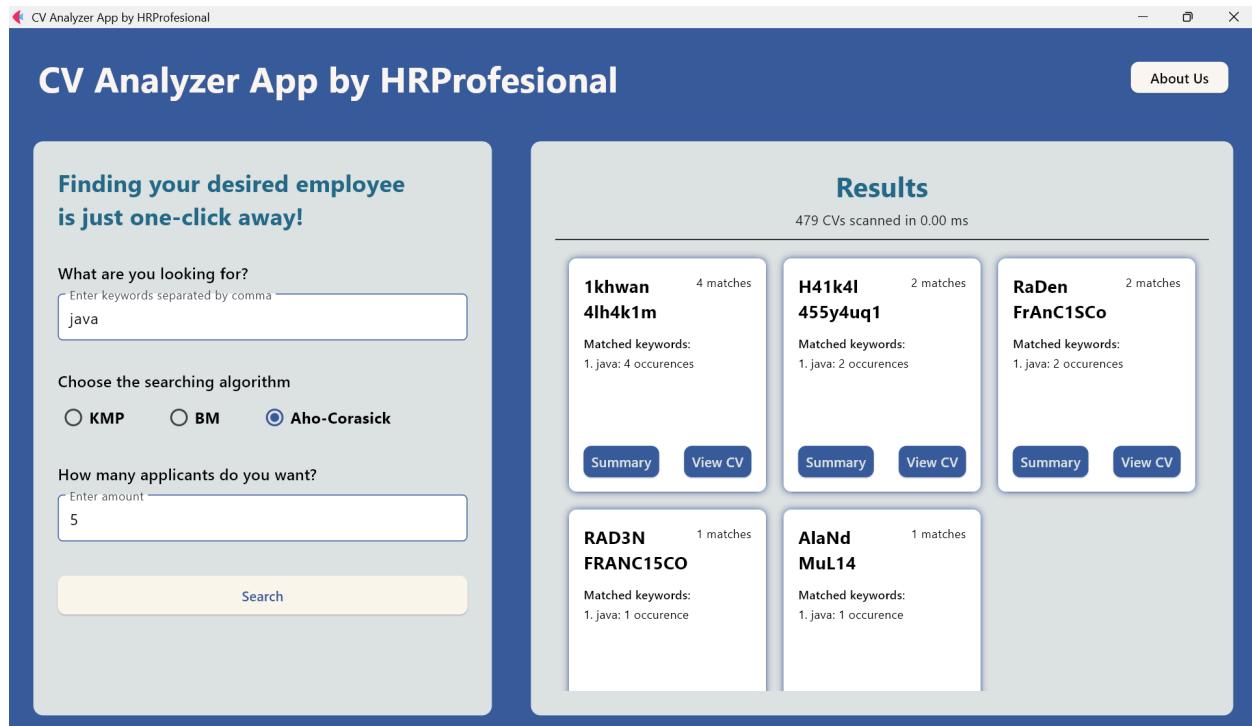
- Algoritma Knuth Morris Pratt (KMP)



- Algoritma Boyer-Moore (BM)

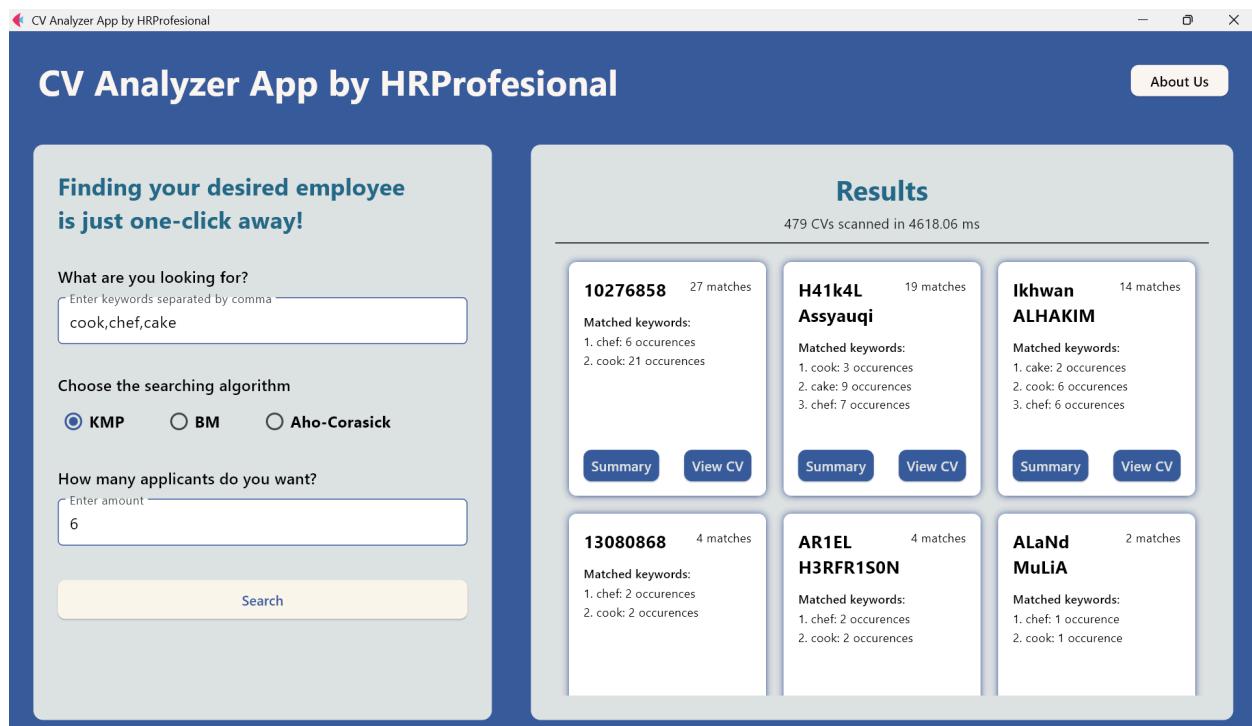


- Algoritma Aho-Corasick

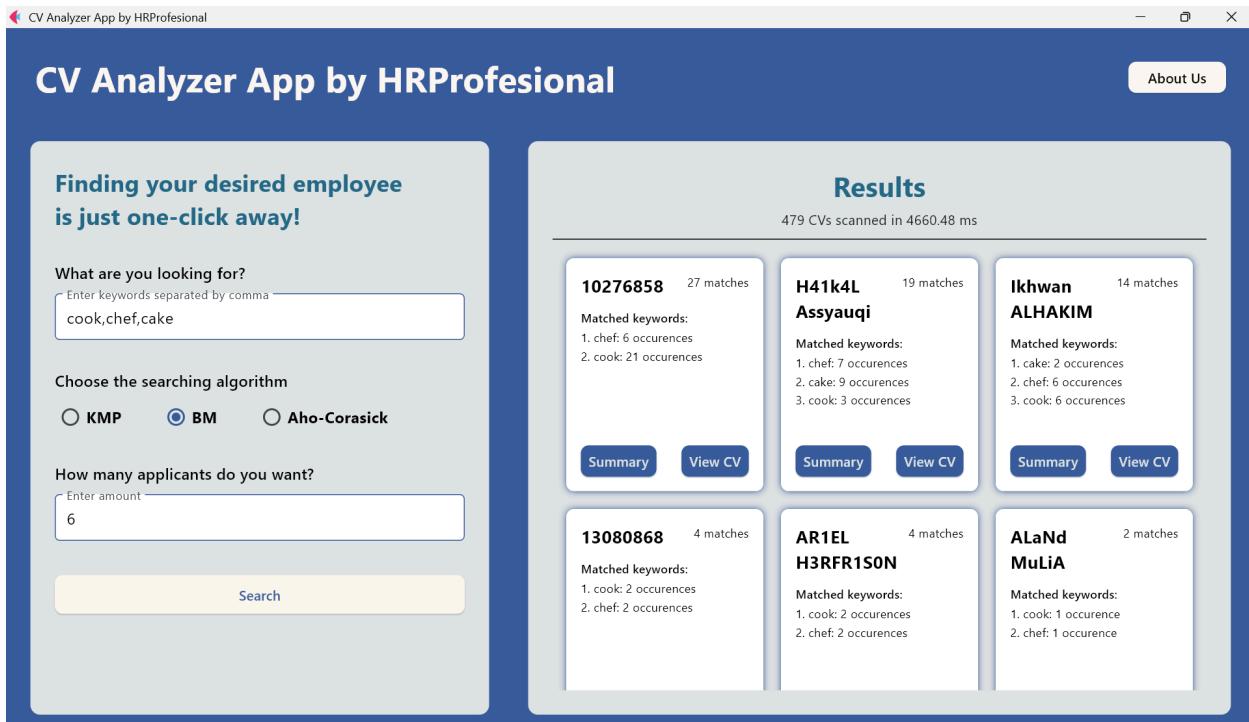


4.3.2 Pencarian beberapa keyword

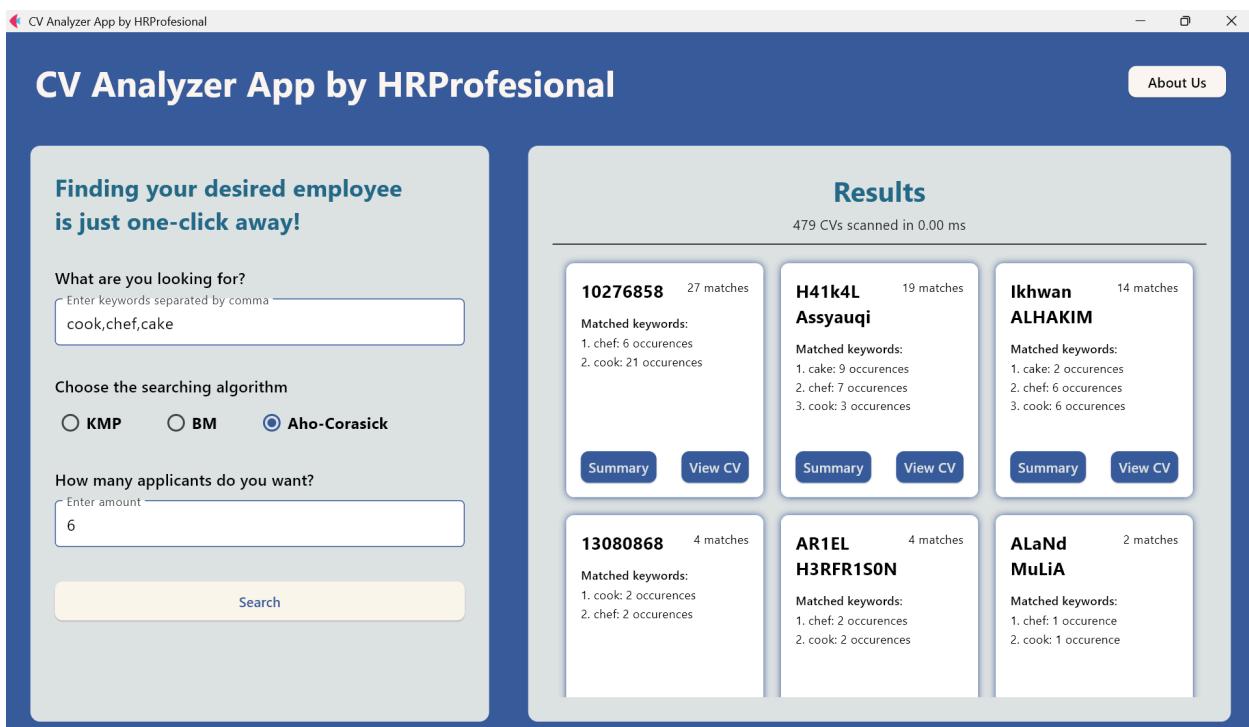
- Algoritma Knuth Morris Pratt (KMP)



- Algoritma Boyer-Moore (BM)



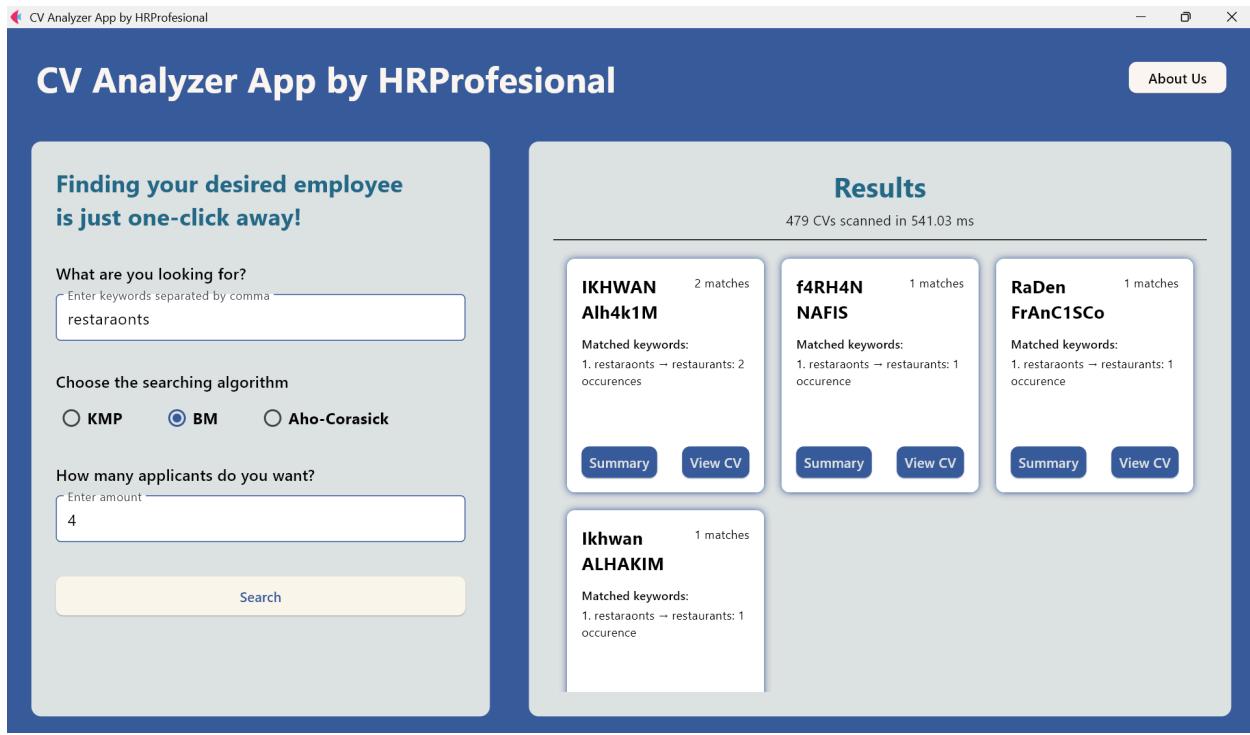
- Algoritma Aho-Corasick



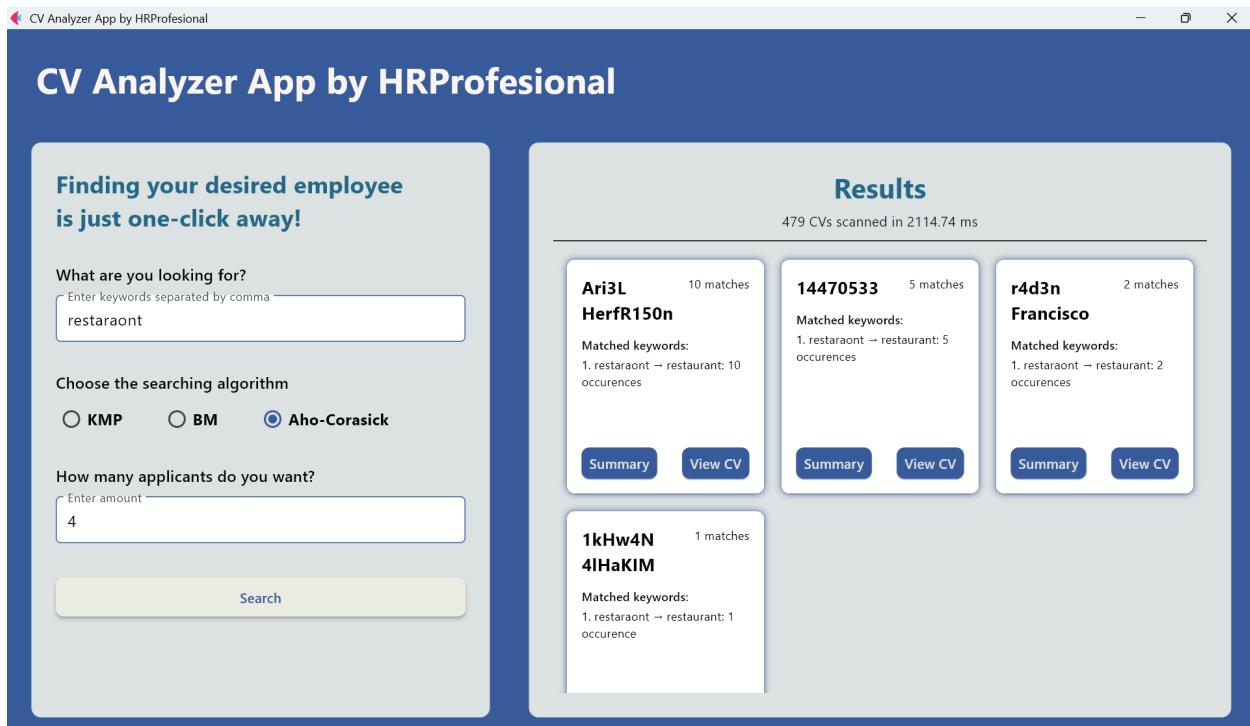
4.3.3 Pencarian 1 keyword dengan saltik

- Algoritma Knuth Morris Pratt (KMP)

- Algoritma Boyer-Moore (BM)

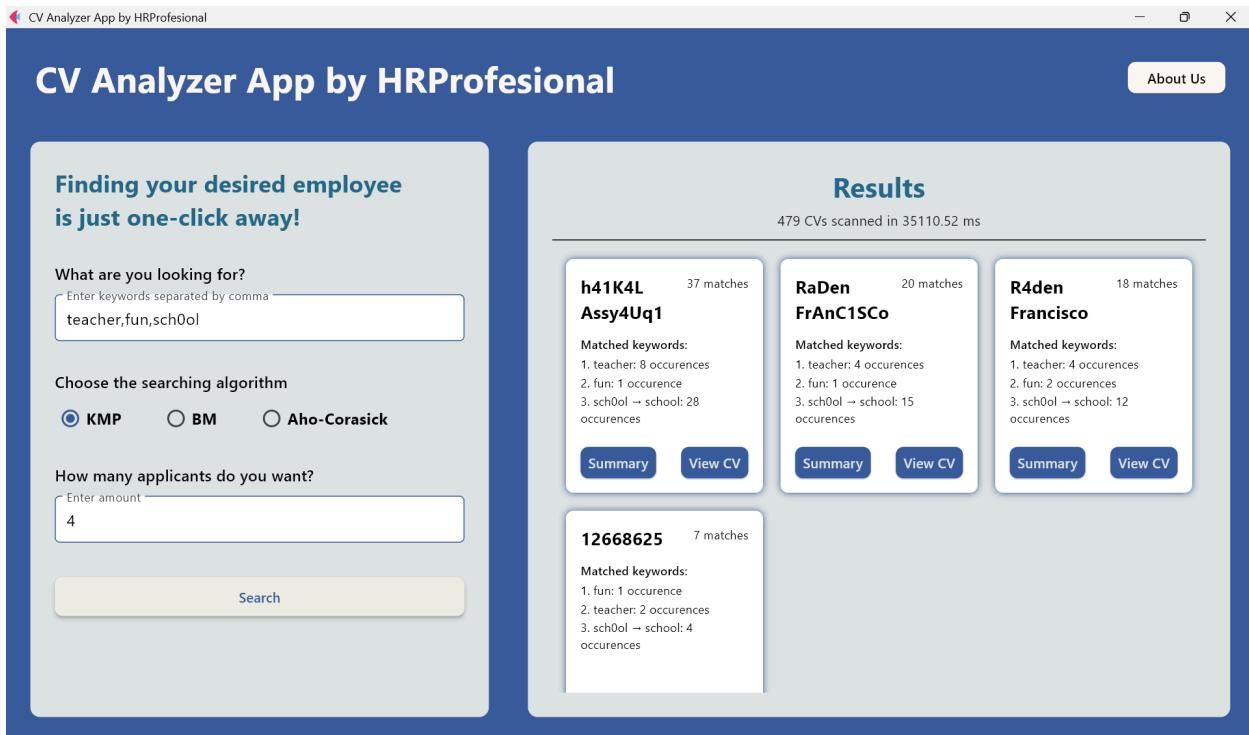


- Algoritma Aho-Corasick

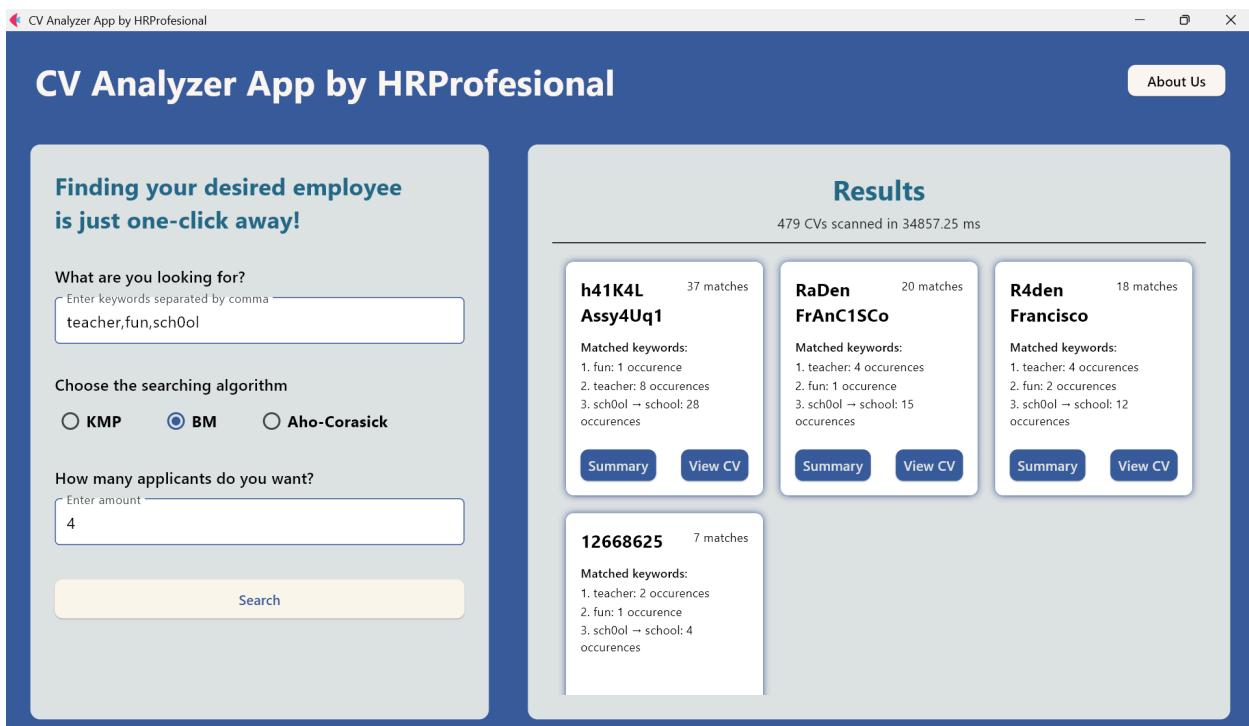


4.2.4 Pencarian beberapa keyword dengan saltik

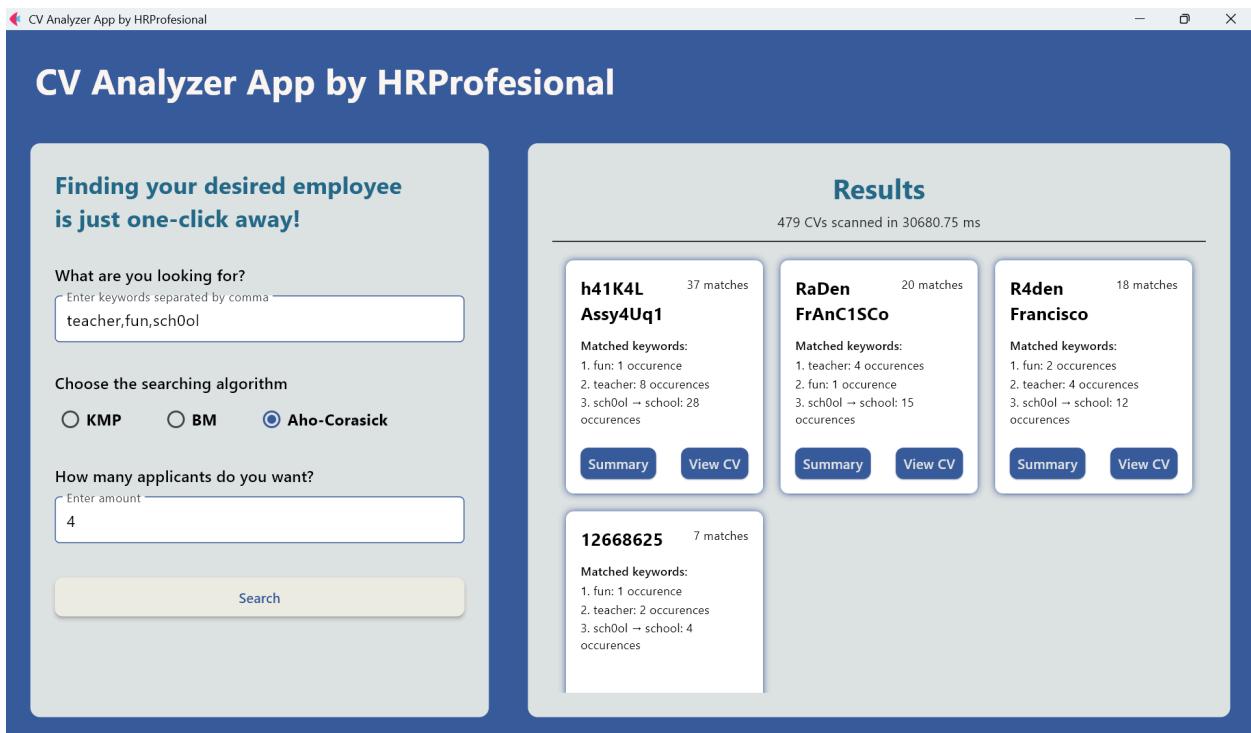
- Algoritma Knuth Morris Pratt (KMP)



- Algoritma Boyer-Moore (BM)



- Algoritma Aho-Corasick



4.3.3 Hasil Pengujian

Berdasarkan hasil pengujian yang telah dilakukan terhadap ketiga algoritma pencocokan string, yaitu Aho-Corasick, Knuth-Morris-Pratt (KMP), dan Boyer-Moore (BM), diperoleh temuan bahwa algoritma Aho-Corasick secara konsisten memberikan waktu pencarian tercepat dibandingkan dengan dua algoritma lainnya dengan rekord pencarian tercepat 0.00 ms. Ini menunjukkan bahwa algoritma Aho-Corasick ini adalah algoritma yang sangat efisien baik dalam pencarian dengan 1 keyword maupun beberapa keyword. Algoritma Knuth Morris Pratt dan Boyer-Moore menghasilkan pencarian yang kurang lebih sama cepatnya. Namun, bila dilihat dari rata-rata waktu pencarian secara keseluruhan, algoritma KMP cenderung sedikit lebih unggul dibandingkan BM. Hal ini dapat disebabkan oleh pola yang berada di dalam CV ATS memiliki banyak prefiks berulang sehingga membuat mekanisme pencocokan ulang dalam KMP yang lebih efisien.

BAB V

Kesimpulan, Saran, dan Refleksi

5.1 Kesimpulan

Berdasarkan hasil penggerjaan dan proses pengembangan aplikasi CV analyzer ini, kami semakin memahami bagaimana cara kerja pencocokan string/*pattern* untuk mendapatkan hasil CV yang diinginkan baik *exact match* maupun *fuzzy match*. Kami juga mempelajari lebih dalam dua algoritma utama yang populer digunakan dalam pencocokan string/*pattern* yakni algoritma Knuth-Morris-Pratt (KMP), algoritma Booye-Moore (BM), dan algoritma bonus Aho-Cosirack. Implementasi algoritma tersebut melalui pencocokan string/*pattern* dari *keywords* yang diminta. Selain itu kami juga mempelajari lebih dalam pencocokan string/*pattern* melalui *regular expression* (regex) dan implementasinya melalui ekstraksi *summary* dari CV aplikasi. Pencocokan string/*pattern* ini dioptimalisasi menggunakan *multithreading*. Penggerjaan tugas besar ini juga memberikan pemahaman lebih lanjut akan pengembangan aplikasi yang fungsional dan memberikan *user experience* yang baik.

5.2 Saran

Aplikasi yang kami buat masih jauh dari sempurna. Untuk ke depannya, dapat lebih memperhatikan struktur program, flow program yang lebih jelas, dan juga menyisihkan waktu lebih banyak untuk bug fixing secara total. Selain itu, diharapkan spesifikasi tugas besar yang lebih terstruktur agar menghindari revisi/perbaikan spesifikasi mendekati tanggal pengumpulan tugas besar.

5.3 Komentar

Kami mengucapkan terima kasih kepada asisten-asisten yang bertanggung jawab atas pelaksanaan tugas besar ini dan dosen-dosen mata kuliah IF2211 Strategi Algoritma yang telah memberikan wawasan dan pengetahuan kepada kami saat perkuliahan. Kami juga mengucapkan terima kasih kepada mahasiswa Teknik Informatika ITB lain yang telah memberikan semangat dan dukungan sehingga pada akhirnya tugas besar ini dapat selesai dengan baik.

5.3 Refleksi

Tugas besar ini telah memberikan kami pengalaman yang sangat mengesankan dari senang hingga tekanan. Kami menghadapi berbagai macam kendala seperti alokasi waktu yang kurang efektif akibat kesibukannya masing-masing, masih kurangnya pengalaman dalam membuat aplikasi, dan kinerja dari

kami sendiri yang kurang baik. Tetapi kami juga mendapatkan banyak manfaat dari penggerjaan tugas besar ini: melatih kerjasama yang baik dan mengembangkan kemampuan kami dalam menerapkan materi perkuliahan secara konkret. Untuk kedepannya, diharapkan tugas besar ini bisa menjadi motivasi untuk kami agar kami lebih berusaha baik dalam penggerjaan tugas-tugas berikutnya.

Lampiran

Tautan Repository

Repository GitHub yang menampung seluruh project ini dapat diakses dengan menekan [tulisan ini](#), atau dengan mengunjungi https://github.com/MaheswaraKaindra/Tubes3_HRProfesional.git.

Tautan Video

Video mengenai project ini dapat diakses dengan menekan [tulisan ini](#), atau dengan mengunjungi <https://youtu.be/XjA6LIFr0c8?si=vECTpefL-sYkK21g>.

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.	✓	

Daftar Pustaka

Munir, R. (2025). *Pencocokan string (String/Pattern Matching)*. Bahan kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI-ITB. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/23-Pencocokan-string-(2025).pdf)

Khodra, M. L. (2025). String matching dengan regular expression. Bahan kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, STEI-ITB. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-\(2025\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/24-String-Matching-dengan-Regex-(2025).pdf)

Kostka, K., & Droubay, X. (n.d.). Aho-Corasick algorithm. *CP-Algorithms.* Retrieved June 15, 2025, from https://cp-algorithms.com/string/aho_corasick.html