LAPORAN TUGAS BESAR 03
IF2211 STRATEGI ALGORITMA

PEMANFAATAN PATTERN MATCHING DALAM MEMBANGUN SISTEM DETEKSI INDIVIDU BERBASIS BIOMETRIK MELALUI CITRA SIDIK JARI

Kelompok 40 : Di_KelarinTubes

Anggota :

Farel Winalda 13522047

Ivan Hendrawan Tan 13522111

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Di era digital ini, keamanan data dan akses menjadi semakin penting. Perkembangan teknologi membuka peluang untuk berbagai metode identifikasi yang canggih dan praktis. Beberapa metode umum yang sering digunakan seperti kata sandi atau pin, namun memiliki kelemahan seperti mudah terlupakan atau dicuri. Oleh karena itu, biometrik menjadi alternatif metode akses keamanan yang semakin populer. Salah satu teknologi biometrik yang banyak digunakan adalah identifikasi sidik jari. Sidik jari setiap orang memiliki pola yang unik dan tidak dapat ditiru, sehingga cocok untuk digunakan sebagai identitas individu.

Pattern matching merupakan teknik penting dalam sistem identifikasi sidik jari. Teknik ini digunakan untuk mencocokkan pola sidik jari yang ditangkap dengan pola sidik jari yang terdaftar di database. Algoritma pattern matching yang umum digunakan adalah Bozorth dan Boyer-Moore. Algoritma ini memungkinkan sistem untuk mengenali sidik jari dengan cepat dan akurat, bahkan jika sidik jari yang ditangkap tidak sempurna.

Dengan menggabungkan teknologi identifikasi sidik jari dan pattern matching, dimungkinkan untuk membangun sistem identifikasi biometrik yang aman, handal, dan mudah digunakan. Sistem ini dapat diaplikasikan di berbagai bidang, seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan.

Pada tugas ini, akan diimplementasikan algoritma Boyer-Moore dan Knuth-Morris-Pratt untuk melakukan deteksi sidik jari dengan aplikasi berbasis GUI dan pembuatan API dengan algoritma dalam bahasa pemrograman C# (*C Sharp*) dengan GUI menggunakan UI framework AvaloniaUI dan pembuatan API dengan menggunakan ASP.NET. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari. Database yang digunakan pun berupa PostgreSQL yang dihosting menggunakan Supabase.

BAB II
LANDASAN TEORI

A. Algoritma *Boyer-Moore*

Algoritma *Boyer-Moore* adalah algoritma pencarian substring yang digunakan untuk mencari pola dalam teks. Algoritma ini dirancang oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini mulai mencocokkan karakter dari sebelah kanan pattern dengan memanfaatkan informasi tentang pola yang sedang dicari untuk melompati sebagian besar karakter dalam teks, sehingga mengurangi jumlah perbandingan yang perlu dilakukan.

Algoritma *Boyer-Moore* menggunakan dua buah heuristik yaitu heuristik *last occurrence* (kadang disebut *bad character heuristic*). Heuristik *last occurrence* berfokus pada pencarian keberadaan karakter terakhir dari pattern yang dicari, dan menggeser pattern sejauh mungkin tanpa melewatkan kemungkinan kecocokan.

B. Algoritma *Knuth-Morris-Pratt*

Algoritma *Knuth-Morris-Pratt* adalah algoritma pencarian substring yang dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersama Vaughan R. Pratt pada tahun 1966. Algoritma ini memiliki cara kerja yang hampir mirip dengan algoritma *Brute-Force*, namun pada algoritma ini melakukan pelompatan dengan memproses pola terlebih dahulu untuk menemukan kecocokan prefiks pola dengan pola itu sendiri.

KMP menggunakan tabel pra-proses yang disebut *partial match table* atau *pi table* yang menyimpan informasi tentang sejauh mana bagian dari string pencarian cocok dengan awalannya sendiri. Cara ini memungkinkan algoritma untuk melanjutkan pencarian dari posisi di mana sebelumnya terjadi ketidakcocokan tanpa mengulangi pemindaian karakter yang telah diketahui tidak akan cocok. Dengan menghindari perbandingan ulang yang tidak perlu, KMP secara signifikan mengurangi jumlah perbandingan yang dibutuhkan.

C. Algoritma Regex

Algoritma *Regular Expression*, atau *Regex*, merupakan metode yang digunakan dalam pemrosesan string yang memungkinkan pencarian, penggantian, dan ekstraksi informasi menggunakan pola yang ditentukan oleh ekspresi reguler. Ekspresi reguler adalah urutan karakter yang membentuk pola pencarian, dan dapat diartikan sebagai bahasa untuk mendeskripsikan atau menentukan himpunan string sesuai aturan tertentu. Aturan yang digunakan dalam program ini yaitu untuk mengubah data dalam database yang menggunakan bahasa alay ke Bahasa Indonesia yang baik dan benar. Algoritma Regex digunakan untuk menentukan apakah suatu string cocok dengan pola tertentu.

D. Algoritma *Levenshtein Distance*

Algoritma *Levenshtein Distance*, adalah metode / algoritma untuk mengukur perbedaan antara dua barisan. Secara informal, jarak Levenshtein antara dua kata adalah jumlah minimum perubahan karakter (penyisipan, penghapusan, atau penggantian) yang diperlukan untuk mengubah satu kata menjadi kata lainnya. Algoritma ini ditemukan oleh matematikawan Soviet Vladimir Levenshtein pada tahun 1965. Cara kerja metode ini dengan melakukan iterasi melalui setiap karakter di kedua string. Jika karakter dari kedua string sama, salin nilai diagonal kiri atas ke sel saat ini. Jika karakter berbeda, ambil nilai minimum dari tiga operasi sebelumnya (penghapusan, penyisipan, penggantian) dan tambahkan 1.

E. Algoritma *Longest Common Subsequence (LCS)*

Algoritma *Longest Common Subsequence*, merupakan algoritma Longest Common Subsequence (LCS) adalah algoritma yang digunakan untuk menemukan subsekuens terpanjang yang muncul dalam dua urutan (string) dalam urutan yang sama, tetapi tidak harus berurutan. Dengan kata lain, LCS adalah urutan karakter yang muncul dalam urutan yang sama di kedua string. Algoritma ini menggunakan pendekatan dinamis untuk memastikan bahwa perhitungan dilakukan dengan efisien, meskipun kompleksitas waktu dan ruangnya masih tergantung pada panjang dua string yang dibandingkan.

F. Algoritma *Hamming Distance*

Algoritma *Hamming Distance*, merupakan jarak antara dua string dengan panjang yang sama, adalah banyaknya posisi di kedua string yang berbeda simbol.[1] Dalam kata lain, jarak Hamming mengukur minimum banyaknya subtitusi yang dibutuhkan untuk mengubah satu string menjadi string lain. Dalam konteks yang lebih umum, jarak Hamming adalah salah satu metrik untuk mengukur edit distance antara dua barisan. Jarak ini dinamai dengan nama matematikawan Amerika, Richard Hamming. Algoritma ini biasanya dipakai untuk mengukur perbedaan antara dua string dengan panjang yang sama. Kompleksitas waktu algoritma ini, yaitu O(n), di mana n adalah panjang string.

G. Aplikasi GUI yang Dikembangkan

Aplikasi ini dikembangkan dengan berbasis GUI dan pembuatan API dengan algoritma dalam bahasa pemrograman C# (*C Sharp*) dengan GUI menggunakan UI framework Avalonia UI dan pembuatan API dengan menggunakan ASP.NET. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.

H. Basis Data

Pembuatan aplikasi ini menggunakan basis data PostgreSQL sebagai DBMS. Data yang dimasukkan berupa data random dengan menggunakan faker.js menggunakan javascript dalam melakukan seeding. Akan tetapi, pengimplementasian skema dan juga migrasi pada tabel tetap menggunakan bahasa pemrograman C# .

BAB III
ANALISIS DAN PEMECAHAN MASALAH

A. Langkah - Langkah Pemecahan Masalah

1. Pendefinisian Masalah

Aplikasi yang dikembangkan bertujuan untuk mencari kemiripan sidik jari dalam basis data berdasarkan gambar sidik jari yang diinput. Pengguna dapat memilih untuk menggunakan algoritma *Knuth-Morris-Pratt* (KMP) atau algoritma *Boyer-Moore* (BM) sebagai cara untuk membandingkan gambar sidik jari yang diinput dengan yang ada di database. Hasil luaran yang ditampilkan dapat berupa hasil yang sama persis atau yang memiliki tingkat kemiripan tertentu.

2. Menetapkan Desain Solusi

Framework UI yang digunakan adalah AvaloniaUI. Aplikasi ini menggunakan *database PostgreSQL* untuk menyimpan data-data mengenai nama file sidik jari beserta data diri pemilik sidik jari tersebut. Bentuk tampilan aplikasi hanya terdiri dari MainWindow sebagai halaman utama aplikasi. Page ini berisi suatu form yang berisi tiga buah button yaitu button pilih citra untuk memasukkan gambar sidik jari oleh user, button switch untuk memilih KMP atau BM beserta button search untuk mencari data diri yang memiliki kesamaan atau kemiripan berdasarkan gambar sidik jari yang dimasukkan sebelumnya.

3. Integrasi dengan Backend

Saat *user* menekan tombol Search, maka aplikasi akan melakukan *Fetch API* yang berfungsi untuk mengirimkan permintaan dari *frontend* ke *backend*. Data yang dikirim berupa gambar sidik jari dan jenis algoritma yang dipilih. Di backend akan terdapat *API* endpoint yang akan menerima request dari frontend dan kemudian menerima data yang dikirim, lalu memproses data tersebut berdasarkan algoritma yang dipilih.

4. Pemrosesan dan Pengoptimalan

Gambar yang diinput oleh user akan terlebih dahulu diubah menjadi hitam putih dan diproses menjadi binary string. Untuk menyederhanakan dan mempercepat proses perbandingan antara gambar yang diinput dengan kumpulan gambar sidik jari lainnya, binary string tersebut kemudian diubah ke dalam bentuk ASCII 8 bit string. Selanjutnya, ASCII string tersebut akan dibandingkan dengan kumpulan gambar sidik jari lainnya yang telah diubah juga ke bentuk ASCII 8 bit string dengan metode KMP atau BM berdasarkan pilihan user.

Pertama-tama digunakan perbandingan citra sidik jari yang berukuran ¼ yang dibandingkan juga dengan gambar citra sidik jari dari dataset yang juga berukuran ¼ . Lalu, jika gambar ¼ tidak ditemukan akan digunakan gambar penuh untuk proses yang lebih lama dan dibandingkan juga dengan dataset gambar penuh. Jika tidak ditemukan akan digunakan algoritma Hamming untuk mencocokkan berapa banyak kata yang berbeda jika ukuran gambar sama. Akan tetapi, jika berbeda akan dihitung LCS (*Longest Common Subsequence*) untuk mencari gambar terbaik. Toleransi / tingkat kemiripan yang dijadikan patokan adalah 20% kemiripan. Jika kurang dari 20% maka tidak ditemukan citra sidik jari yang cocok.

5. Respon ke Frontend

Hasil dari perbandingan akan dikirim ke frontend dan ditampilkan dengan sebuah gambar sidik jari yang sama persis atau paling mirip dengan gambar yang diinput oleh user. Selain itu, ada beberapa detail tulisan yang ditampilkan seperti data diri pemilik sidik jari tersebut, lama waktu pencarian, persentase kecocokan antara gambar input dan gambar hasil, dan juga pesan error jika terjadi suatu kesalahan.

6. Enkripsi dan Dekripsi

Dalam memasukkan data yang cukup penting, seperti data identitas dan data pada KTP. Data yang krusial seperti itu, diperlukan enkripsi yang tepat sehingga data-data yang ada dalam database dapat tersimpan dengan aman. Metode enkripsi yang digunakan yaitu dengan melakukan shift kiri sebesar 21, lalu dilakukan operasi XOR dengan key yang sudah ditentukan. Data baru akan di dekripsi untuk mencari kecocokan pencarian.

7. Basis Data

Data yang dimasukkan berupa data yang telah di-enkripsi untuk jenis tabel Biodata. Lalu, tabel SidikJari berisi data dari semua path gambar beserta nama pemilik sidik jari tersebut.

B. Proses Penyelesaian Solusi Dengan Algoritma KMP dan BM

1. Pemanggilan Fungsi Backend Algoritma KMP

Elemen:

- ASCII string input : Gambar yang diinput user dan di convert menjadi ASCII 8 bit string
- ASCII string perbandingan: Gambar yang akan dibandingkan dan di convert menjadi ASCII 8 bit string
- Tujuan : Menemukan string input dan perbandingan yang sama persis atau memiliki tingkat persentase tertentu
- Type struct data: Menyimpan data diri pemilik sidik jari hasil perbandingan
- Integer bestMatchPercentage : Menyimpan nilai kemiripan terbaik hasil perbandingan

Proses:

1. Ubah gambar yang diinput menjadi hitam putih, lalu diubah menjadi binary string dan diubah lagi menjadi ASCII 8 bit string.
2. Lakukan perbandingan string dari proses nomor 1 dengan kumpulan gambar sidik jari lainnya yang diubah juga seperti nomor 1.
3. Jika ditemukan hasil yang sama persis, maka proses perbandingan akan dihentikan dan akan mengembalikan struct data yang berisi data diri pemilik sidik jari hasil perbandingan beserta tingkat kemiripan sebesar 100%.

4. Jika tidak ditemukan hasil yang sama persis, maka perbandingan akan menghasilkan gambar yang memiliki tingkat persentase kemiripan tertinggi berdasarkan algoritma Hamming Distance atau LCS Proses ini juga akan mengembalikan struct data yang berisi data diri pemilik sidik jari hasil perbandingan beserta tingkat kemiripannya.
5. Jika tidak ditemukan sebuah hasil, maka akan mengembalikan sebuah pesan yang berisi “No matching data found”.

2. Pemanggilan Fungsi Backend Algoritma BM

Elemen:

- ASCII string input : Gambar yang diinput user dan di convert menjadi ASCII 8 bit string
- ASCII string perbandingan: Gambar yang akan dibandingkan dan di convert menjadi ASCII 8 bit string
- Tujuan : Menemukan string input dan perbandingan yang sama persis atau memiliki tingkat persentase tertentu
- Type struct data: Menyimpan data diri pemilik sidik jari hasil perbandingan
- Integer bestMatchPercentage : Menyimpan nilai kemiripan terbaik hasil perbandingan

Proses:

1. Ubah gambar yang diinput menjadi hitam putih, lalu diubah menjadi binary string dan diubah lagi menjadi ASCII 8 bit string.
2. Lakukan perbandingan string dari proses nomor 1 dengan kumpulan gambar sidik jari lainnya yang diubah juga seperti nomor 1.

3. Jika ditemukan hasil yang sama persis, maka proses perbandingan akan dihentikan dan akan mengembalikan struct data yang berisi data diri pemilik sidik jari hasil perbandingan beserta tingkat kemiripan sebesar 100%.
4. Jika tidak ditemukan hasil yang sama persis, maka perbandingan akan menghasilkan gambar yang memiliki tingkat persentase kemiripan tertinggi berdasarkan algoritma Hamming Distance atau LCS Proses ini juga akan mengembalikan struct data yang berisi data diri pemilik sidik jari hasil perbandingan beserta tingkat kemiripannya.
5. Jika tidak ditemukan sebuah hasil, maka akan mengembalikan sebuah pesan yang berisi "No matching data found".

C. Fitur Fungsional dan Arsitektur Aplikasi

1. Fitur Fungsional

Ada beberapa fitur fungsional yang terdapat pada aplikasi ini, antara lain:

- Tombol pilih citra untuk memasukkan gambar sidik jari yang mau dibandingkan
- Tombol switch untuk memilih antara algoritma KMP atau BM
- Tombol search untuk mengirim request ke backend untuk mencari hasil jalur terpendek dari judul awal ke judul tujuan.

2. Arsitektur Aplikasi

```
.
├── src/
│   ├── client/
│   │   ├── Assets/
│   │   ├── bin/
│   │   ├── obj/
│   │   ├── App.axaml
```

```
│   │   ├── App.axaml.cs
│   │   ├── app.manifest
│   │   ├── client.csproj
│   │   ├── MainWindow.axaml
│   │   ├── MainWindow.axaml.cs
│   │   └── Program.cs
│   └── server/
│       ├── bin/
│       ├── Controllers/
│       │   ├── BiodataController.cs
│       │   ├── BioSearchController.cs
│       │   ├── BMController.cs
│       │   ├── BoyerMoore.cs
│       │   ├── Hamming.cs
│       │   ├── ImageConverter.cs
│       │   ├── KMPController.cs
│       │   ├── KnuthMorrisPratt.cs
│       │   ├── Levenshtein.cs
│       │   ├── LongestCommonSubsequence.cs
│       │   └── SidikJariController.cs
│       ├── Migrations/
│       ├── models/
│       ├── obj/
│       ├── Properties/
│       ├── appsettings.Development
│       ├── appsettings.json
│       ├── DatabaseContext.cs
```

```
│       ├── Private.cs
│       ├── Program.cs
│       └── server.csproj
└── README.md
```

Folder *source code* dibagi menjadi 2 yaitu *client / frontend* dan *server / backend*. Pada folder *client* dibagi menjadi beberapa file dan folder penting, yaitu:

1. Folder Asset : Folder ini berisi berbagai asset resource yang digunakan seperti gambar-gambar untuk styling
2. File client.csproj : File ini berisi semua dependency dan package yang digunakan dalam projek frontend ini.
3. File MainWindow.axaml : File ini berisi tag-tag untuk axaml yang digunakan dalam men-design tampilan utama GUI
4. File MainWindow.axaml.cs : File ini merupakan Controller dari File axaml yang berisikan metode-metode serta beberapa fungsi untuk memunculkan komponen di dalam GUI
5. File Program.cs : File ini merupakan loader untuk GUI

Pada folder *server* dibagi menjadi beberapa file juga, yaitu:

1. File Program.cs : File ini merupakan loader utama dalam pembuatan API beberapa konfigurasi dicantumkan di sini
2. File DatabaseContext.cs : File ini merupakan file utama yang digunakan untuk melakukan Migrasi database dengan memasukkan model-model yang dijadikan tabel untuk dimasukkan ke dalam database. Dalam kasus ini, terdapat pembuatan 3 tabel, yaitu: Tabel Biodata, Tabel SidikJari, dan Tabel EncryptedBiodata.
3. Folder Migrations : Folder ini berisi file-file migrasi yang sudah di-translate yang berisi riwayat-riwayat penambahan tabel dan modifikasi tabel.
4. File Private.cs : File ini berisikan 2 Class, yaitu: Encrypt dan Decrypt. File ini berisikan metode yang dilakukan dalam pembuatan Enkripsi dan Dekripsi dari sebuah string dengan key yang telah ditentukan.

5. Folder models : Folder ini berisi semua jenis tipe dan struktur data yang telah saya cantumkan sebelumnya. Berisi file-file deklarasi tipe-tipe seperti: Biodata, Request / Result, SidikJari, StringRequest / StringResult.
6. File appsettings.json : File ini berisi konfigurasi untuk melakukan koneksi menuju database
7. File server.csproj : Sama seperti client.csproj.cs, file ini berisi semua dependency dan package yang digunakan dalam projek backend.
8. Folder Controllers : Folder ini berisikan beberapa controller yang digunakan baik dalam pembuatan endpoint maupun dalam pembuatan algoritma yang digunakan dalam endpoint. Folder controller ini berisikan beberapa file yaitu:
    a. File BiodataController.cs : File ini merupakan file konfigurasi untuk endpoint "/api/biodata" yang memiliki Method GET untuk menampilkan semua data dari Biodata yang ada dalam database. Data ini berisikan data yang telah dienkripsi
    b. File SidikJariController.cs : File ini merupakan file konfigurasi untuk endpoint "/api/sidikjari" yang memiliki Method GET untuk menampilkan semua data dari sidikjari yang ada dalam database.
    c. File BioSearchController.cs : File ini merupakan file konfigurasi untuk endpoint "/api/biosearch" yang memiliki method POST yang digunakan untuk membandingkan kata-kata aneh dengan kata normal. Lalu mengembalikan hasil berupa StringResult yang berisi informasi tentang biodata yang paling mendekati data yang akan dicari
    d. File BMController.cs : File ini merupakan file konfigurasi untuk endpoint "/api/bm" yang memiliki method POST yang digunakan untuk mencocokan citra sidik jari yang ingin dicari dengan yang tersedia di database dan dataset dengan algoritma Boyer Moore. Lalu mengembalikan SearchResult berupa sidikjari yang paling cocok dan juga persentase kemiripan tersebut.
    e. File KMPController.cs : File ini merupakan file konfigurasi untuk endpoint "/api/bm" yang memiliki method POST yang digunakan untuk mencocokan citra sidik jari yang ingin dicari dengan yang tersedia di

database dan dataset dengan algoritma Knuth Morris Pratt. Lalu mengembalikan SearchResult berupa sidikjari yang paling cocok dan juga persentase kemiripan tersebut.

f. File BoyerMoore.cs : File ini merupakan file yang berisi algoritma dari Boyer Moore sendiri yang digunakan oleh kontroler yang lain.
g. File KnuthMorrisPratt.cs : File ini merupakan file yang berisi algoritma dari Knuth Morris Pratt sendiri yang digunakan oleh kontroler yang lain.
h. File Levenshtein.cs : File ini merupakan file yang berisi algoritma dari LevenShtein sendiri yang digunakan oleh kontroler yang lain.
i. File LongestCommonSubsequence.cs : File ini merupakan file yang berisi algoritma dari LCS sendiri yang digunakan oleh kontroler yang lain.
j. File ImageConverter.cs : File ini merupakan file yang berisi metode dalam pengambilan gambar citra sidikjari yang terbaik dan juga metode pengubahan ukuran dan pengubahan data baik teks binari, teks ascii, dan juga byte.

D. Contoh Ilustrasi Kasus

Salah satu contoh kasus yang dapat diselesaikan dengan aplikasi yang mengimplementasikan algoritma KMP dan BM adalah mencari gambar sidik jari yang sama atau mirip berdasarkan gambar yang diinput oleh user dengan kumpulan gambar yang akan dibandingkan.

Mula-mula, gambar yang diinput oleh user akan diubah menjadi hitam putih terlebih dahulu, kemudian gambar tersebut diubah bentuknya menjadi binary string, binary string ini akan diubah lebih lanjut menjadi ASCII 8 bit string untuk menyederhanakan proses perbandingan antara string hasil input dengan string hasil kumpulan gambar.

Dengan algoritma KMP, string dari gambar yang diinput akan dibandingkan dengan setiap gambar sidik jari yang telah diubah ke dalam bentuk ASCII 8 bit string dengan metode KMP. Jika algoritma KMP mengembalikan hasil *true*, maka proses perbandingan akan langsung dihentikan dan akan mengembalikan identitas diri mengenai pemilik sidik jari, lama waktu pencarian, dan persentase tingkat kemiripan. Jika tidak ada hasil yang ditemukan dengan KMP,

maka proses akan dilanjutkan dengan metode Hamming Distance dengan syarat kedua buah ASCII string memiliki panjang yang sama. Jika panjang berbeda, maka akan menggunakan metode LCS (Longest Common Subsequence). Lalu, untuk pengecekan biodata melalui nama dilakukan Algoritma Regex terlebih dahulu untuk mengubah huruf-huruf besar menjadi kecil, menghapus spasi lalu dengan mengubah kata-kata alay angka menjadi huruf. Akan digunakan algoritma KMP untuk pengecekan string nama yang sudah dinormalisasi. Jika tidak mengeluarkan hasil, maka akan digunakan Levenshtein untuk menghitung kemiripan kedua string berdasarkan persentase kemiripan tertinggi.

Proses dengan algoritma BM mirip seperti dengan algoritma KMP, yang membedakan hanya cara kerja dari membandingkan kedua buah ASCII string. Dengan algoritma BM, string dari gambar yang diinput akan dibandingkan dengan setiap gambar sidik jari yang telah diubah ke dalam bentuk ASCII 8 bit string dengan metode BM. Jika algoritma BM mengembalikan hasil *true*, maka proses perbandingan akan langsung dihentikan dan akan mengembalikan identitas diri mengenai pemilik sidik jari, lama waktu pencarian, dan persentase tingkat kemiripan. Jika tidak ada hasil yang ditemukan dengan BM, maka proses akan dilanjutkan dengan metode Hamming Distance dengan syarat kedua buah ASCII string memiliki panjang yang sama. Jika panjang berbeda, maka akan menggunakan metode LCS (Longest Common Subsequence). Lalu, untuk pengecekan biodata melalui nama dilakukan Algoritma Regex terlebih dahulu untuk mengubah huruf-huruf besar menjadi kecil, menghapus spasi lalu dengan mengubah kata-kata alay angka menjadi huruf. Akan digunakan algoritma BM untuk pengecekan string nama yang sudah dinormalisasi. Jika tidak mengeluarkan hasil, maka akan digunakan Levenshtein untuk menghitung kemiripan kedua string berdasarkan persentase kemiripan tertinggi.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi Teknis Program

Program dibagi menjadi berbagai entitas yang dapat saling dihubungkan untuk mendukung kinerja pembuatan tugas besar ini. Terdapat folder Models yang berfungsi sebagai deklarasi tipe untuk berbagai tipe data buatan. Berbagai tipe data buatan yang digunakan yaitu :

1. Biodata

   Atribut dari kelas ini:

   a. NIK (string)
   b. NamaAlay (string)
   c. TempatLahir (string)
   d. TanggalLahir (DateTime)
   e. JenisKelamin (string)
   f. Golongan Darah (string)
   g. Alamat (string)
   h. Agama (string)
   i. StatusPerkawinan (string)
   j. Pekerjaan (string)
   k. Kewarganegaraan (string)

2. EncryptedBiodata

   Atribut dari kelas ini:

   a. NIK (string)
   b. NamaAlay (string)
   c. TempatLahir (string)
   d. TanggalLahir (string)
   e. JenisKelamin (string)
   f. Golongan Darah (string)
   g. Alamat (string)
   h. Agama (string)

    i. StatusPerkawinan (string)
    j. Pekerjaan (string)
    k. Kewarganegaraan (string)
3. SidikJari

   Atribut dari kelas ini:
    a. BerkasCitra (string)
    b. Nama (string)
4. SearchRequest

   Atribut dari kelas ini:
    a. Data (string)
5. SearchResult

   Atribut dari kelas ini:
    a. SidikJari (SidikJari)
    b. MatchPercentage (int)
6. StringRequest

   Atribut dari kelas ini:
    a. RealName (string)
    b. IsKMP (bool)
7. StringResult

   Atribut dari kelas ini:
    a. Biodata (Biodata)

Dari struktur folder yang telah diberikan di atas, terdapat fitur / fungsi utama yang dibangun dari program keseluruhan ini yang dibagi menjadi beberapa kelas, yaitu:

1. BiodataController.cs : Berisi konfigurasi API untuk mendapat semua biodata enkripsi.

   Adapun beberapa fungsi, yaitu:
    - Fungsi IEnumerable<EncryptedBiodata> Get() : mengembalikan list biodata yang dienkripsi
2. BioSearchController.cs : Berisi konfigurasi API untuk mencari biodata yang paling cocok

   Adapun beberapa fungsi, yaitu:

- Fungsi string NormalizeName(string name) : mengembalikan nama yang telah dinormalisasi dengan menggunakan algoritma regex.
- Fungsi List<string> NormalizeAndSplitName(string name) : mengembalikan list nama yang telah dinormalisasi dengan menggunakan algoritma regex.
- Fungsi Post (request) : memproses request yang berisi nama normal yang dibandingkan dengan semua nama jelek yang dienkripsi. Melewati tahap dekripsi terlebih dahulu, lalu menggunakan algoritma BM atau KMP. Jika tidak ditemukan akan digunakan levenshtein lalu diambil kemiripan yang paling baik. Mengembalikan hasil biodata bersih yang terbaik.

3. BMController.cs : Berisi konfigurasi API untuk mencari citra sidik jari dengan algoritma Boyer Moore

   Adapun beberapa fungsi, yaitu:

   - Fungsi Task<IActionResult> Post([FromBody] SearchRequest request) : memproses request yang berisi format file base64 dari gambar. Lalu diubah menjadi binary dan disederhanakan menjadi ASCII. Pencarian menggunakan algoritma BM lalu jika tidak ditemukan akan menggunakan Algoritma Hamming terlebih dahulu dan jika Hamming tidak melebihi 40%, maka akan digunakan algoritma LCS. Lalu, mengembalikan Response berupa sidikjari paling cocok beserta persen kemiripannya
4. KMPController.cs : Berisi konfigurasi API untuk mencari citra sidik jari dengan algoritma Knuth Morris Pratt.

   Adapun beberapa fungsi, yaitu:

   - Fungsi Task<IActionResult> Post([FromBody] SearchRequest request) : memproses request yang berisi format file base64 dari gambar. Lalu diubah menjadi binary dan disederhanakan menjadi ASCII. Pencarian menggunakan algoritma KMP lalu jika tidak ditemukan akan menggunakan Algoritma Hamming terlebih dahulu dan jika Hamming tidak melebihi 40%, maka akan digunakan algoritma LCS. Lalu, mengembalikan Response berupa sidikjari paling cocok beserta persen kemiripannya
5. BoyerMoore.cs : Berisi implementasi fungsi algoritma Boyer Moore.

   Adapun beberapa fungsi, yaitu:

- Fungsi bool BMSearch(string text, string pattern) : Fungsi ini mencari apakah pola (pattern) ada dalam teks (text). Pertama, jika panjang pola lebih panjang dari teks, pola dan teks ditukar. Kemudian, tabel "last occurrence" dibangun menggunakan fungsi BuildLast. Algoritma ini memulai pencarian dari kanan ke kiri, mulai dari indeks i yang ditetapkan pada posisi akhir pola dalam teks. Jika karakter pada indeks i dalam teks cocok dengan karakter pada indeks j dalam pola, pencocokan terus dilakukan ke kiri. Jika terjadi ketidakcocokan, algoritma menggunakan tabel "last occurrence" untuk menentukan seberapa jauh pola dapat digeser ke kanan. Proses ini berulang hingga seluruh teks telah diperiksa atau pola ditemukan. Jika pola ditemukan, fungsi mengembalikan true, jika tidak, fungsi mengembalikan false
- Fungsi Dictionary<char, int> BuildLast(string pattern) : fungsi ini digunakan untuk membangun tabel "last occurrence" (kejadian terakhir) dari karakter-karakter dalam sebuah pola (pattern). Tabel ini menyimpan posisi terakhir di mana setiap karakter muncul dalam pola

6. KnuthMorrisPratt.cs : Berisi implementasi fungsi algoritma Knuth Morris Pratt.
   Adapun beberapa fungsi, yaitu:
   - Fungsi bool KMPSearch(string text, string pattern) : Fungsi ini mencari apakah pola (pattern) terdapat dalam teks (text). Fungsi ini pertama-tama memastikan bahwa pola lebih pendek atau sama panjangnya dengan teks, kemudian membangun tabel "border" dengan ComputeBorder untuk pola tersebut. Algoritma ini melakukan pencocokan karakter demi karakter dari teks dan pola. Jika karakter cocok, pencocokan dilanjutkan; jika semua karakter pola cocok, fungsi mengembalikan true. Jika terjadi ketidakcocokan, fungsi menggunakan tabel "border" untuk menggeser pola tanpa memeriksa ulang karakter yang sudah cocok, sehingga meningkatkan efisiensi pencarian. Jika pencarian selesai tanpa menemukan pola, fungsi mengembalikan false
   - Fungsi int[] ComputeBorder(string pattern) : Fungsi ComputeBorder membangun tabel "border" atau "failure function" untuk pola yang diberikan, yang digunakan dalam algoritma pencocokan string Knuth-Morris-Pratt (KMP). Fungsi ini menginisialisasi sebuah array b dengan panjang yang sama dengan pola, yang

akan menyimpan panjang perbatasan terpanjang untuk setiap prefix pola. Iterasi dilakukan melalui karakter-karakter pola, membandingkan setiap karakter dengan karakter sebelumnya. Jika karakter saat ini cocok dengan karakter sebelumnya, panjang perbatasan diperbarui dan indeks bergerak maju. Jika tidak cocok, fungsi menggunakan nilai sebelumnya dari tabel perbatasan untuk menghindari perbandingan ulang

7. Levenshtein.cs : Berisi implementasi algoritma Levenshtein Distance
   Adapun beberapa fungsi, yaitu:
   - Fungsi int Compute(string s, string t) : Fungsi ini menghitung jarak antara dua string, s dan t, yang merupakan jumlah minimum operasi (penyisipan, penghapusan, atau penggantian) yang diperlukan untuk mengubah satu string menjadi string lainnya. Fungsi ini menggunakan matriks dua dimensi d untuk melacak biaya operasi di setiap langkah, menginisialisasi baris dan kolom pertama dengan indeks mereka masing-masing, lalu mengisi matriks berdasarkan perbandingan karakter dari kedua string. Hasil akhirnya adalah nilai di sel paling bawah dari matriks, yang merupakan jarak Levenshtein antara kedua string tersebut
8. LongestCommonSubsequence.cs : Berisi implementasi algoritma LCS
   Adapun beberapa fungsi, yaitu:
   - Fungsi int Compute(string text, string pattern) : Fungsi ini menghitung panjang subsekuens umum terpanjang (Longest Common Subsequence, LCS) antara dua string, text dan pattern. Fungsi ini menggunakan dua array satu dimensi prev dan curr untuk melacak panjang LCS saat iterasi melalui karakter-karakter dari kedua string, menghindari penggunaan matriks dua dimensi untuk menghemat memori. Dengan membandingkan karakter dari kedua string dan memperbarui array berdasarkan hasil perbandingan, fungsi ini akhirnya mengembalikan panjang LCS yang disimpan dalam elemen terakhir dari array prev
9. Hamming.cs : Berisi implementasi algoritma Hamming Distance
   Adapun beberapa fungsi, yaitu:
   - Fungsi int HammingDist(string text, string pattern) : Fungsi ini menghitung jarak Hamming antara dua string, text dan pattern, dengan mengiterasi melalui setiap

karakter dari kedua string dan menghitung jumlah posisi di mana karakter-karakternya berbeda. Jika panjang kedua string berbeda, fungsi akan mengalami kesalahan karena mengasumsikan kedua string memiliki panjang yang sama. Hasil akhirnya adalah jumlah perbedaan karakter antara kedua string, yang disimpan dalam variabel count dan dikembalikan sebagai hasil fungsi

10. ImageConverter.cs : Berisi fungsi-fungsi dalam mengubah dan mengolah gambar

    Adapun beberapa fungsi, yaitu:

    - Fungsi string ConvertByteToAsciiString(byte[] data) : Fungsi ini mengubah byte menjadi string dalam format ASCII
    - Fungsi string ConvertBinaryStringToAscii(string binaryString) : Fungsi ini mengubah string Binary menjadi string dalam format ASCII
    - Fungsi string FindBestPixelSegment(byte[] imagePixels, int imageWidth, int imageHeight, int segmentWidth) : Fungsi ini mencari segmen terbaik dari sebuah gambar yang memiliki proporsi piksel hitam dan putih paling mendekati 50:50 dalam setiap barisnya. Fungsi ini menghitung perbedaan rasio hitam dan putih untuk setiap segmen dalam setiap baris gambar, dan memilih segmen dengan perbedaan rasio terkecil. Segmen terbaik ini kemudian dikonversi menjadi string ASCII dan dikembalikan sebagai hasil fungsi.
    - Fungsi (byte[], int, int) ConvertImageToGrayscaleByteArray(byte[] imageData, int scaleFactor = 1) : Fungsi ini mengonversi gambar berwarna yang diberikan dalam bentuk array byte menjadi gambar grayscale dalam bentuk array byte dengan ukuran yang dapat diskalakan. Gambar dimuat dan diubah ukurannya sesuai dengan faktor skala yang diberikan, kemudian dikonversi menjadi grayscale. Hasil konversi disimpan dalam bentuk BMP ke dalam memori, diambil sebagai array byte, dan akhirnya dikembalikan bersama dengan lebar dan tinggi gambar yang telah diubah ukurannya.

11. Private.cs : Berisi fungsi-fungsi dalam meng-enkripsi dan deskripsi sebuah string

    Adapun beberapa fungsi

    - Fungsi string GetEncrypt(string data) : Fungsi ini mengenkripsi string input menggunakan kombinasi operasi shift dan XOR dengan kunci yang telah ditentukan. Data binary dilakukan shift kiri sebesar 21, setelah itu hasilnya

di-XOR dengan byte dari kunci yang berulang. Byte hasil enkripsi ini kemudian dikonversi menjadi string base64.

- Fungsi string GetDecrypt(string encryptedData) : Fungsi ini mendekripsi string terenkripsi yang dikodekan dalam format Base64. Byte terenkripsi pertama-tama dikonversi kembali dari Base64 ke array byte, kemudian setiap byte di-XOR dengan key. Setelah itu, dilakukan shift kanan sebesar 21. Akhirnya, array byte yang didekripsi ini dikonversi kembali menjadi string UTF-8 dan dikembalikan.

12. SidikJariController.cs : Berisi konfigurasi API untuk mendapat semua biodata enkripsi Adapun beberapa fungsi
    - Fungsi IEnumerable<SidikJari> Get() : mengembalikan list sidik jari

B. Penjelasan Tata Cara Penggunaan Program

1. Silahkan lakukan clone *repository* ini dengan cara menjalankan perintah berikut pada terminal

```
git clone https://github.com/Bodleh/Tubes3_Di_KelarinTubes.git
```

2. Jalankan perintah berikut pada terminal untuk memasuki root directory program

```
cd Tubes3_Di_KelarinTubes
```

3. Setelah pengguna berada pada root directory, buatlah sebuah terminal baru
4. Dari terminal pertama, jalankan perintah-perintah berikut

```
cd src/client
dotnet build
dotnet run
```

5. Dari terminal kedua, jalankan perintah berikut

```
cd src/server
```

```
dotnet run
```

6. Memasukkan dataset dengan membuat folder test/data (https://drive.google.com/drive/folders/1KvJcSkQXgjz4lb1Ayg71KQB5zrFIPNOG?usp=sharing)

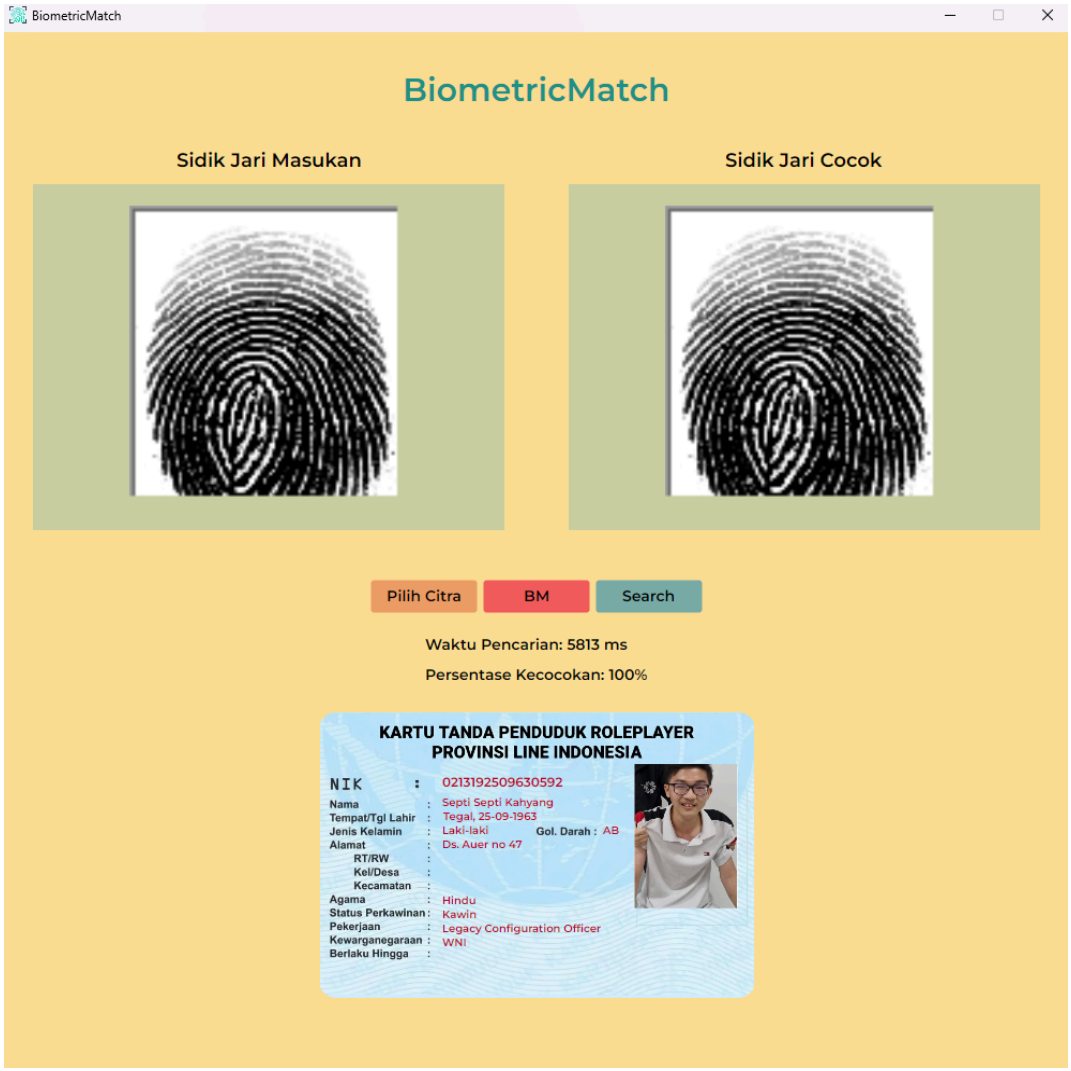
```
mkdir test
```

```
mkdir data
```

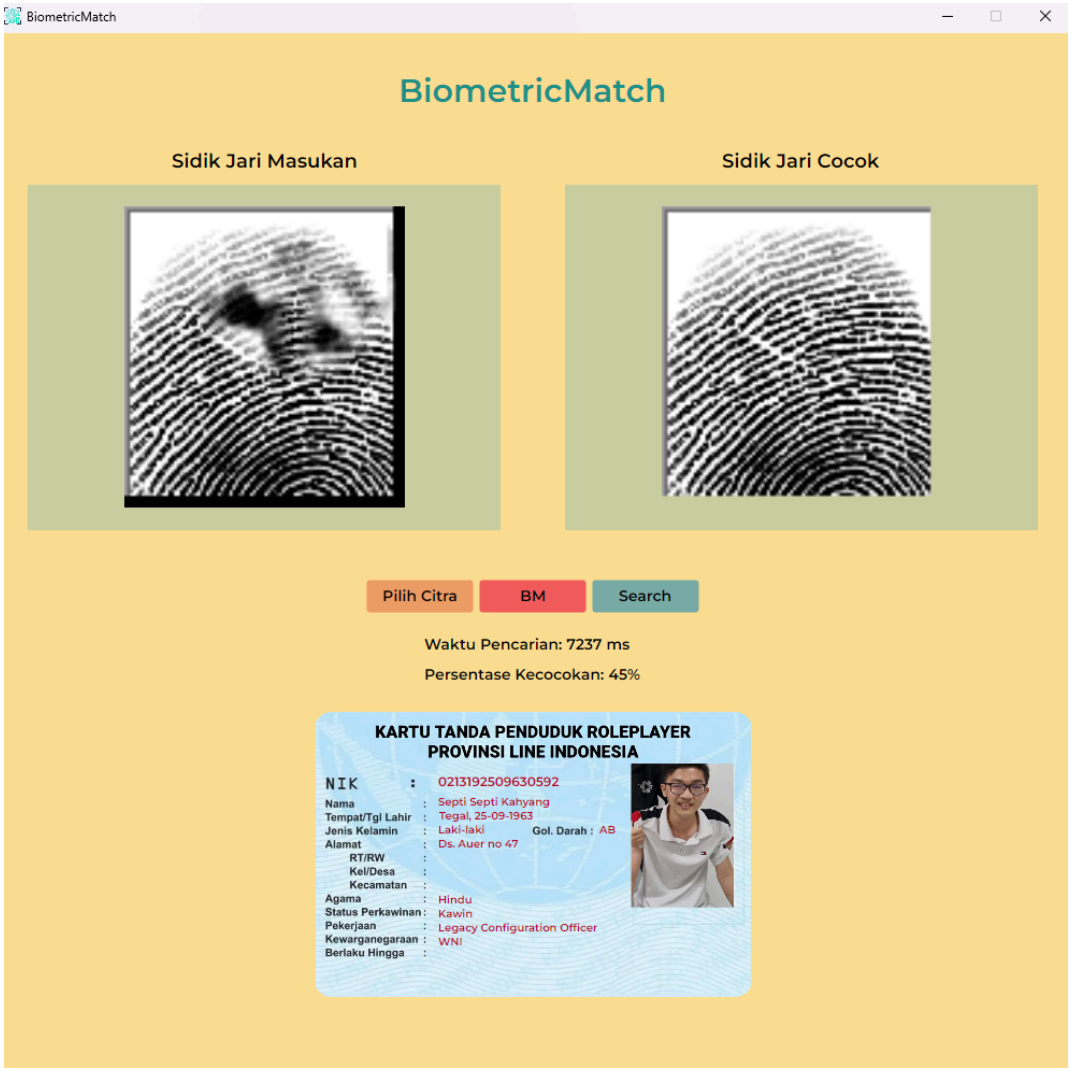
7. Setelah laman utama aplikasi muncul, maka aplikasi dapat digunakan

C. Hasil Pengujian

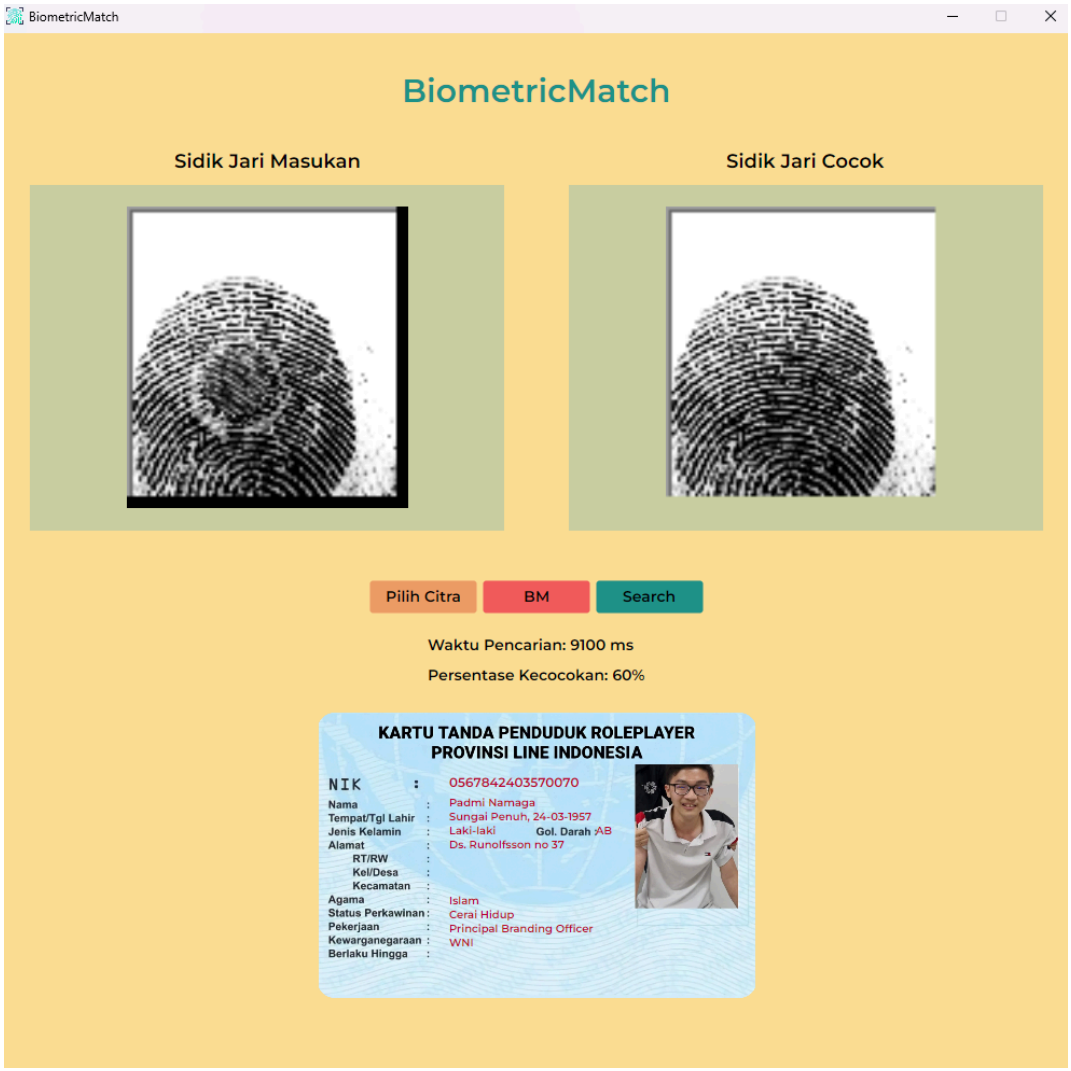
1. Hasil Pengujian Boyer-Moore (BM) :
    - Boyer-Moore : Real



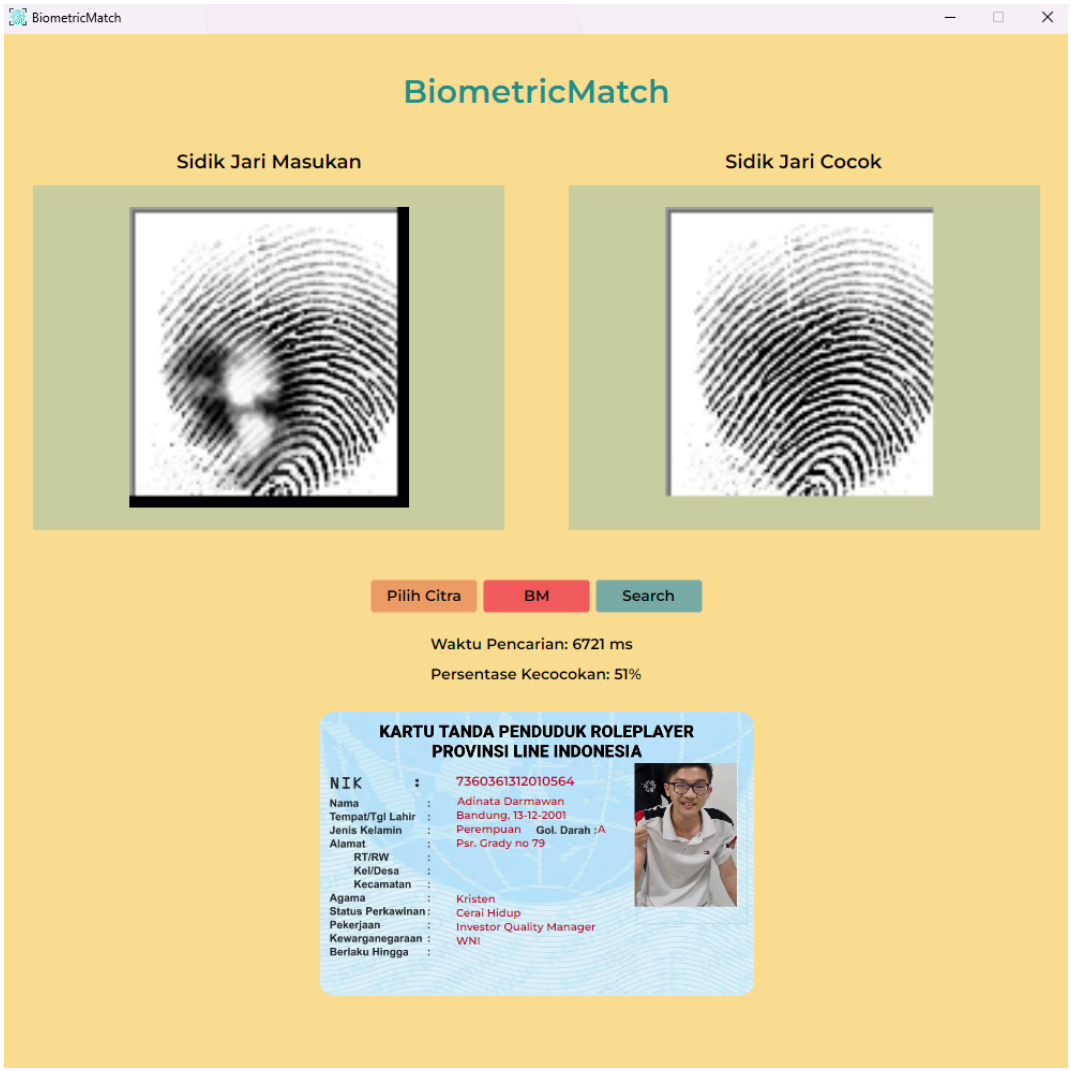
- Boyer-Moore : Altered Easy



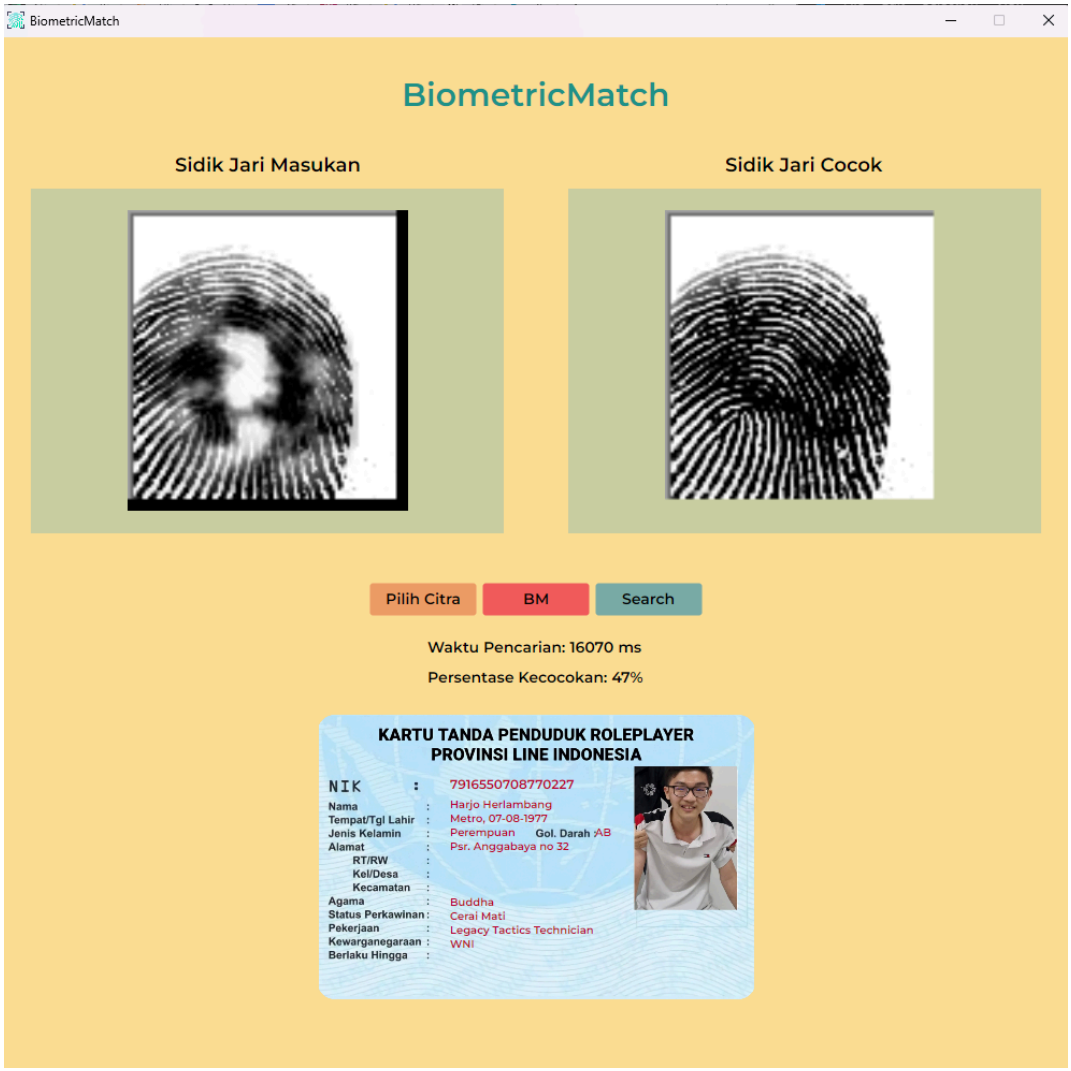
- Boyer-Moore : Altered Easy



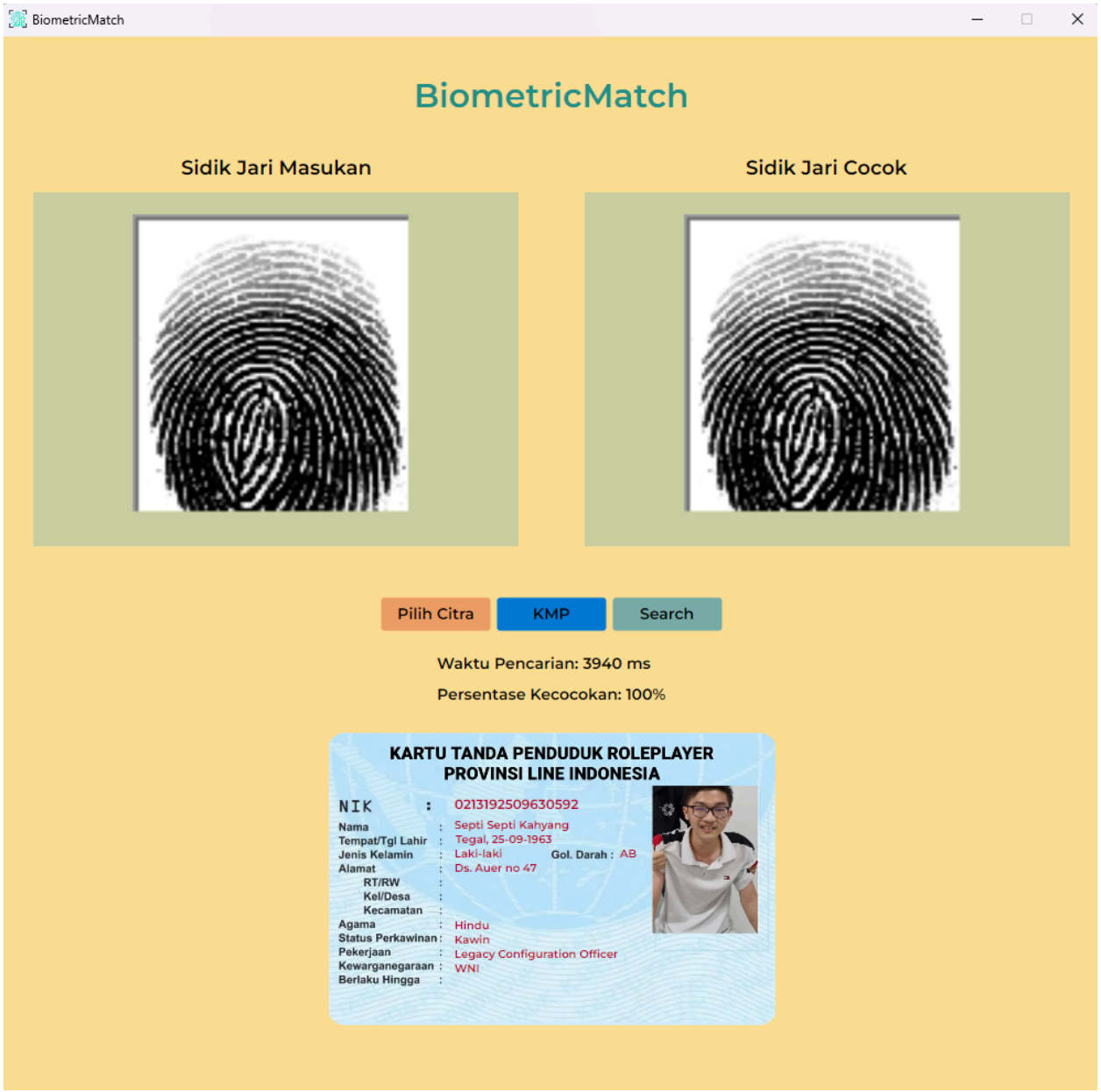
- Boyer-Moore : Altered Medium



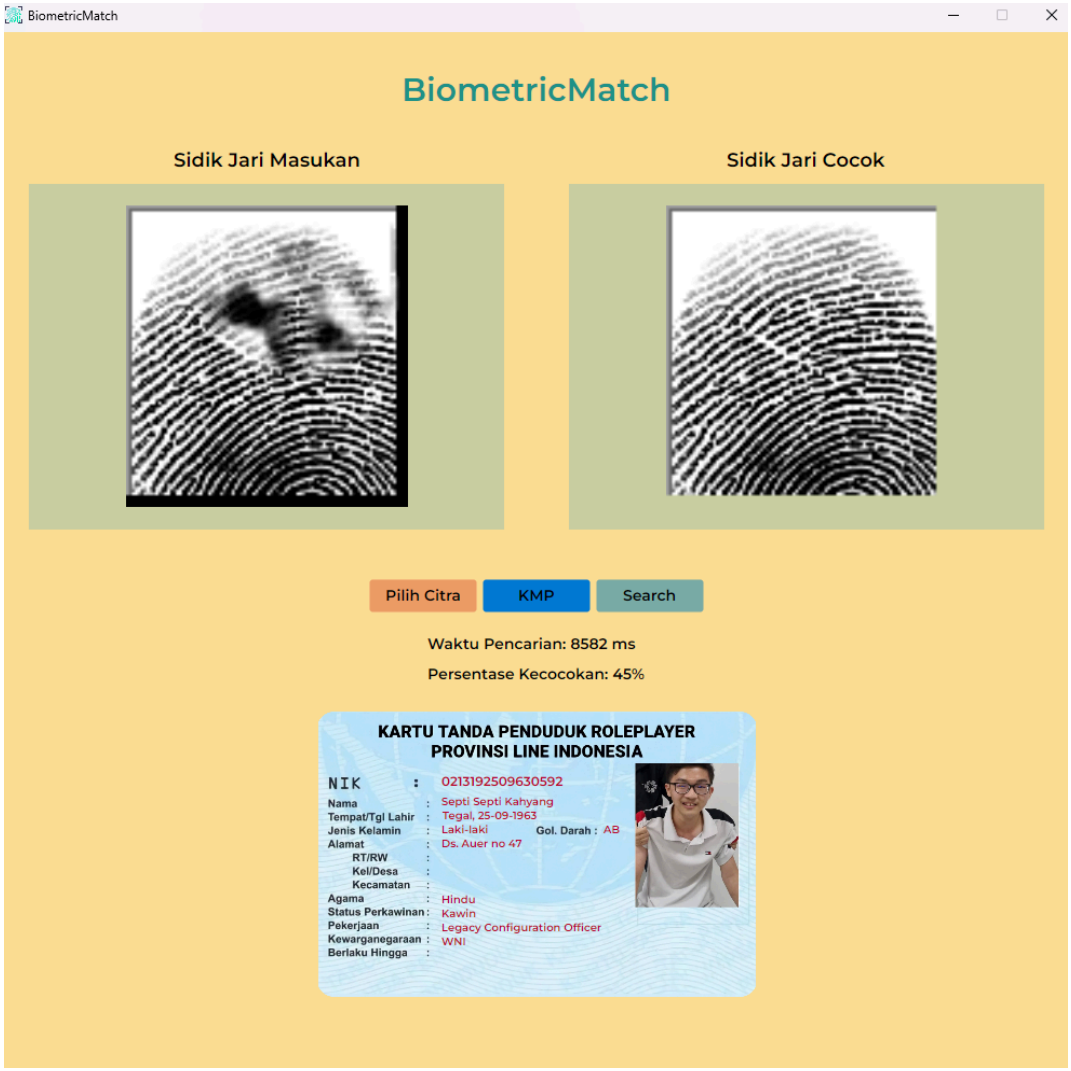
- Boyer-Moore : Altered Hard



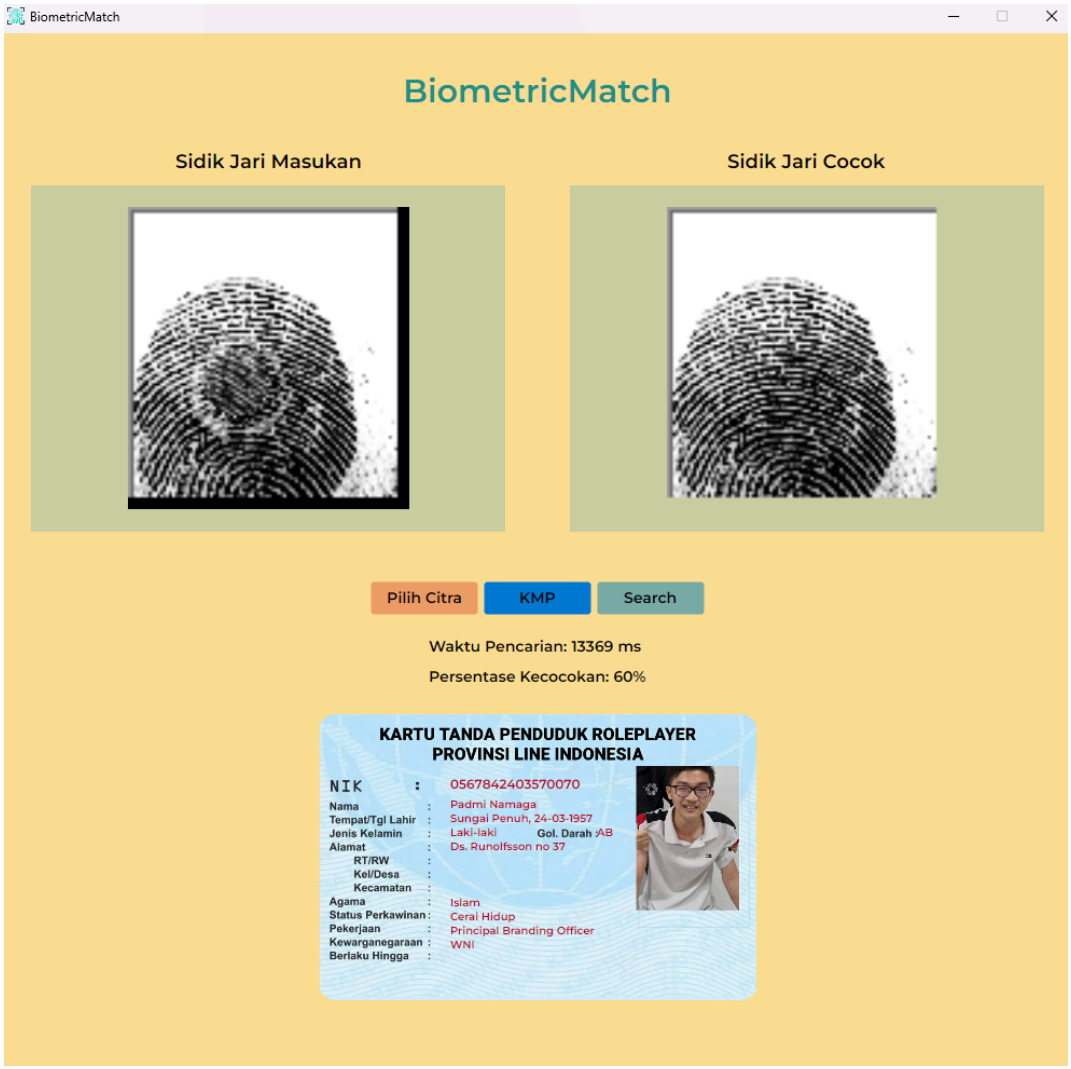
2. Hasil Pengujian Knuth-Morris-Pratt (KMP) :
    - Knuth-Morris-Pratt : Real



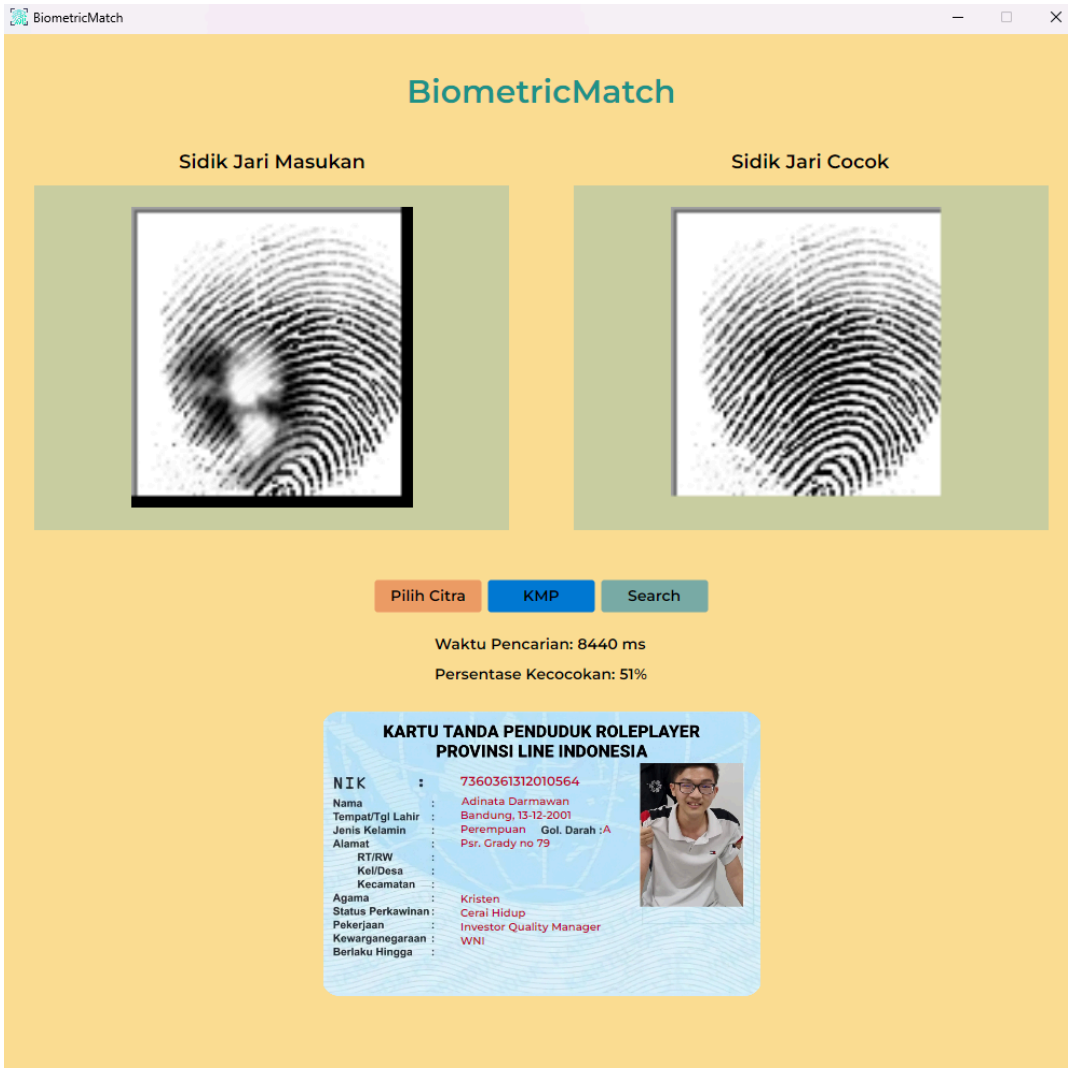
- Knuth-Morris-Pratt : Altered Easy



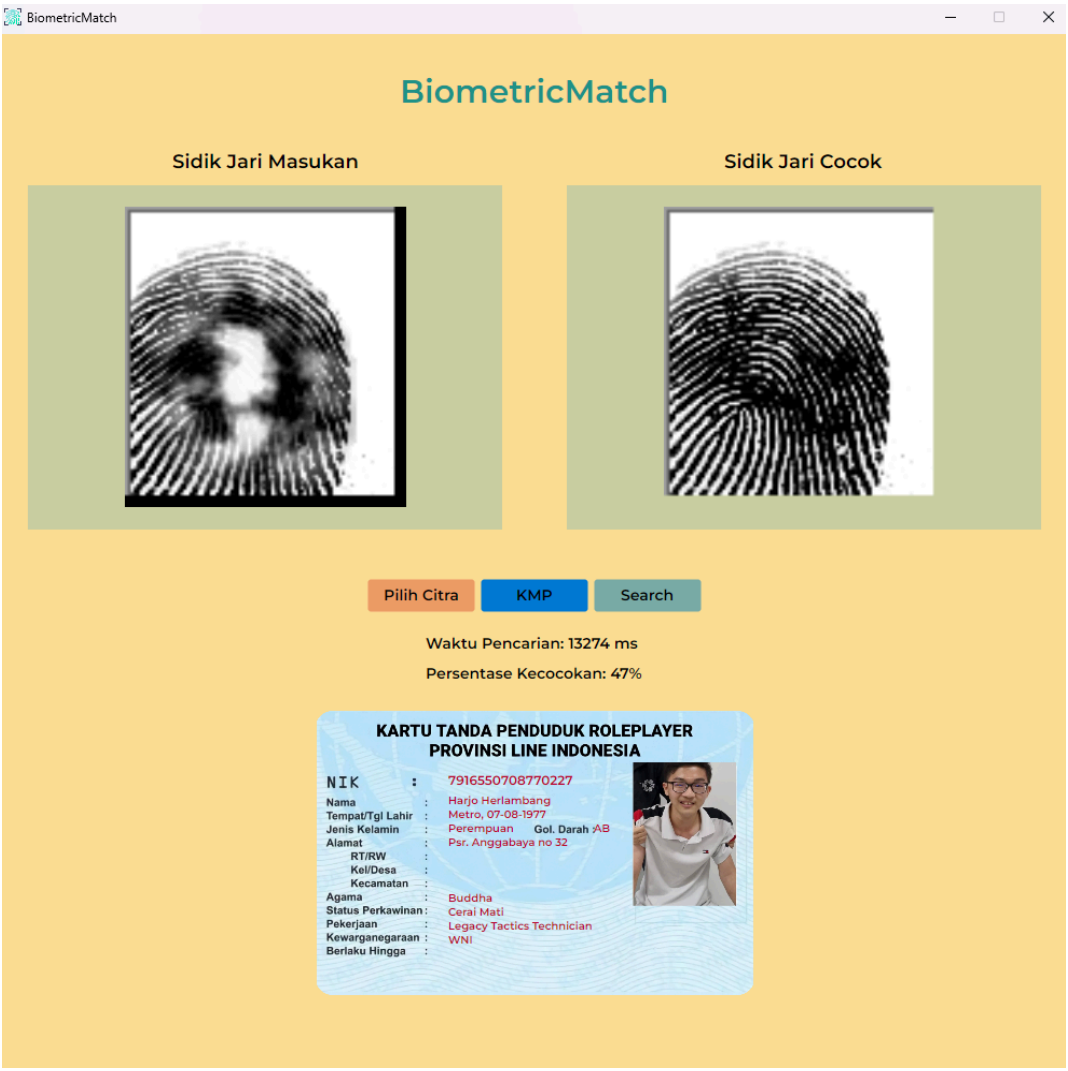
- Knuth-Morris-Pratt : Altered Easy



- Knuth-Morris-Pratt : Altered Medium



- Knuth-Morris-Pratt : Altered Hard



D. Analisis Hasil Pengujian

Algoritma KMP adalah salah satu metode dalam pencocokan string yang dirancang untuk mengurangi jumlah perbandingan yang tidak perlu dengan memanfaatkan tabel *border function* sebagai informasi tambahan. Tabel *border function* memungkinkan proses pencarian untuk melompati sub kalimat yang tidak cocok tanpa perlu memeriksa ulang karakter-karakter dalam sub kalimat tersebut. Tabel ini dibuat berdasarkan pola dari kalimat itu sendiri, dengan mencatat indeks di mana pencarian dapat dilanjutkan setelah ketidakcocokan terjadi. Hal ini membantu mengurangi waktu dalam pencocokan, terutama dalam kasus-kasus di mana teks mengandung banyak pengulangan.

Mengenai kompleksitas algoritma dari KMP, pada kasus terbaik terjadi dimana ketika karakter pertama dari pola tidak cocok dengan karakter di teks, atau setiap karakter di teks cocok

sempurna dengan pola. Kompleksitas waktu dalam kasus terbaik ini adalah O(N) dengan N adalah panjang kalimat. Akan tetapi, kasus terburuk dapat terjadi ketika pola ditemukan hampir di akhir teks atau setiap kegagalan pencocokan terjadi di akhir pola kalimat sehingga kompleksitas waktu pada kasus terburuk ini adalah O(N + M) dengan N adalah panjang kalimat dan M adalah panjang pola.

Pada algoritma BM, algoritma akan memaksimalkan jumlah karakter yang dapat diloncati selama pencocokan dengan memanfaatkan informasi heuristik yang dimiliki oleh algoritma BM sehingga lebih unggul daripada KMP pada kalimat yang panjang atau pola yang panjang. Informasi heuristik ini didapat dari tabel *bad character* yang berisi tentang seberapa jauh pola harus digeser ketika terjadi ketidakcocokan huruf.

Kasus terbaik dalam algoritma BM terjadi ketika pola tidak ditemukan dalam teks dan heuristik *bad character* memungkinkan algoritma untuk melompati sebagian besar teks, kompleksitas waktu algoritma ini adalah O(N/M) dimana N adalah panjang kalimat dan M adalah panjang polanya.

Ketika perbandingan pola dengan KMP atau BM tidak menghasilkan kecocokan yang 100%, Hamming Distance dan Longest Common Subsequence (LCS) dapat menjadi solusi lain dalam melakukan perbandingan pola dengan kalimat.

Algoritma Hamming Distance digunakan untuk mengukur seberapa berbeda dua string dengan panjang yang sama. Algoritma ini menghitung jumlah posisi di mana dua string tersebut memiliki karakter yang berbeda. Sedangkan algoritma Longest Common Subsequence adalah algoritma yang digunakan untuk menemukan subsekuensi terpanjang yang muncul secara berurutan dalam dua string atau lebih. Tidak seperti subsekuensi biasa, subsekuensi terpanjang tidak harus berada pada posisi yang berurutan dalam string asli, tetapi urutan karakter harus tetap sama.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Dalam tugas besar ini, kami telah berhasil menerapkan Pemanfaatan Pattern Matching dalam Membangun Sistem Deteksi Individu Berbasis Biometrik Melalui Citra Sidik Jari dengan menggunakan algoritma pattern matching seperti Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), Levenshtein Distance, Hamming Distance, dan LCS untuk mengatasi kasus di mana algoritma KMP atau BM tidak berhasil menemukan kesamaan.

Tugas besar ini memperlihatkan bahwa algoritma pattern matching dapat digunakan dengan efektif untuk pengembangan sistem deteksi individu berbasis biometrik berupa sidik jari. Untuk pengembangan lebih lanjut, seperti implementasi integrasi algoritma enkripsi yang lebih kuat untuk memenuhi tantangan keamanan siber masa depan. Selain itu, penelitian lebih lanjut dapat dilakukan dalam melakukan optimasi algoritma dan teknologi sensor sidik jari dalam meningkatkan akurasi gambar sidik jari.

B. Saran

Secara menyeluruh, algoritma KMP atau BM telah memenuhi kebutuhan dasar dalam proses membandingkan dua buah string. Maka dari itu, kedua algoritma ini tidak memerlukan perbaikan lebih lanjut. Namun, jika tidak ditemukan sebuah hasil dengan algoritma KMP atau BM, maka diperlukan sebuah proses tambahan dengan algoritma Hamming Distance atau LCS yang memerlukan optimalisasi lebih lanjut karena proses ini memakan waktu yang lebih lama dalam melakukan perbandingan. Optimalisasi yang dapat dilakukan berupa pemrosesan paralel untuk mempersingkat proses perbandingan.

C. Tanggapan dan Refleksi

Kami merasa bahwa tugas besar kali ini cukup mudah dan juga cukup susah. Penerapan algoritma KMP dan BM yang sudah umum sehingga terdapat banyak contoh referensi dan penjelasannya di dunia internet. Akan tetapi, algoritma Hamming Distance atau LCS yang

diterapkan dalam aplikasi ini cukup menyulitkan karena hasil luaran yang diberikan terkadang tidak sesuai yang diharapkan.

Di sisi lain, kami dapat menambah wawasan baru dari pembuatan aplikasi GUI ini yang menggunakan bahasa C#, ekosistem .NET, dan juga berbagai framework yang memperlancar perjalanan tugas ini.

Akhir kata, tugas besar ini memberikan banyak manfaat seperti menambah pemahaman kami tentang algoritma pencocokan pola terutama string dalam kasus ini dan juga tentang pengembanganaplikasi GUI berbasis C#. Kami merasa bahwa pengalaman ini sangat berharga dan akan bermanfaat dalam waktu ke depan.

LAMPIRAN

Pranala repository :

https://github.com/Bodleh/Tubes3_Di_KelarinTubes

Pranala database:

postgres://postgres.imvavpfgesfoaantzvnk:[YOUR-PASSWORD]@aws-0-ap-southeast-1.pooler.supabase.com:6543/postgres

Pranala dataset yang digunakan:

https://www.kaggle.com/datasets/ruizgara/socofing

DAFTAR PUSTAKA

Munir, Rinaldi. n.d. “Pencocokan string (String matching/pattern matching)” Informatika. Diakses 24 Mei 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Munir, Rinaldi. n.d. “Pencocokan string dengan Regular Expression (Regex)” Informatika. Diakses 24 Mei 2024. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf