

LAPORAN TUGAS BESAR 3 IF2211

STRATEGI ALGORITMA

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



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S. T, M.Sc
Asisten pembimbing : Mohammad Rifqi Farhansyah

Disusun oleh:
Kelas 02 - Kelompok 7 - HapHipHop

Marzuli Suhada M (13522070)
Ahmad Mudabbir Arif (13522072)
Naufal Adnan (13522116)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

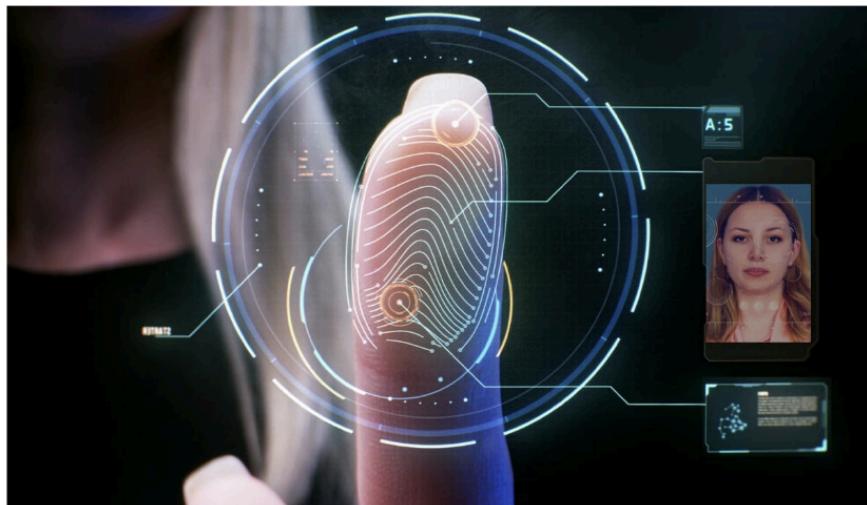
DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	3
BAB II.....	5
2.1 Algoritma KMP (Knuth-Morris-Pratt).....	5
2.2 Algoritma BM (Boyer-Moore).....	6
2.3 Regex (Regular Expression).....	8
2.4 Teknik Pengukuran Kemiripan Persentase.....	8
2.5 Pembangunan Aplikasi Desktop.....	10
BAB III.....	12
3.1 Langkah-Langkah Pemecahan Masalah.....	12
3.1.1 Menentukan Citra Sidik Jari.....	12
3.1.2 Menentukan Area yang Cocok.....	12
3.1.3 Konversi Citra Sidik Jari ke Binary.....	12
3.1.4 Konversi Binari ke ASCII 8 Bits.....	12
3.1.5 Pencocokan Sidik Jari.....	13
3.1.6 Pencarian identitas pemilik sidik jari.....	13
3.2 Proses penyelesaian solusi dengan algoritma KMP dan BM.....	14
3.2.1 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma KMP.....	14
3.2.2 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma BM.....	14
3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop yang Dibangun.....	15
3.3.1 Fitur Fungsional.....	15
3.3.2 Arsitektur Aplikasi Desktop.....	16
3.4 Ilustrasi Kasus.....	18
BAB IV.....	22
4.1 Spesifikasi Teknis Program.....	22
4.1.1 Struktur Data.....	45
4.1.2 Fungsi dan Prosedur.....	46
4.2 Tata Cara Penggunaan Program.....	50
4.3 Hasil Pengujian.....	52
4.3.1 Test Case 1.....	52
4.3.2 Test Case 2.....	53
4.3.3 Test Case 3.....	54
4.3.4 Test Case 4.....	55
4.4 Analisis Hasil Pengujian.....	56
4.5 Penjelasan Implementasi Bonus Enkripsi Data.....	57
BAB V.....	60
5.1 Kesimpulan.....	60
5.2 Saran.....	60
5.3 Refleksi.....	60

5.4 Tanggapan.....	61
DAFTAR PUSTAKA.....	62
Repository.....	63
Youtube.....	63

BAB I

DESKRIPSI TUGAS



Gambar 1. Ilustrasi fingerprint recognition pada deteksi berbasis biometrik.

Sumber: <https://www.aratek.co/news/unlocking-the-secrets-of-fingerprint-recognition>

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.

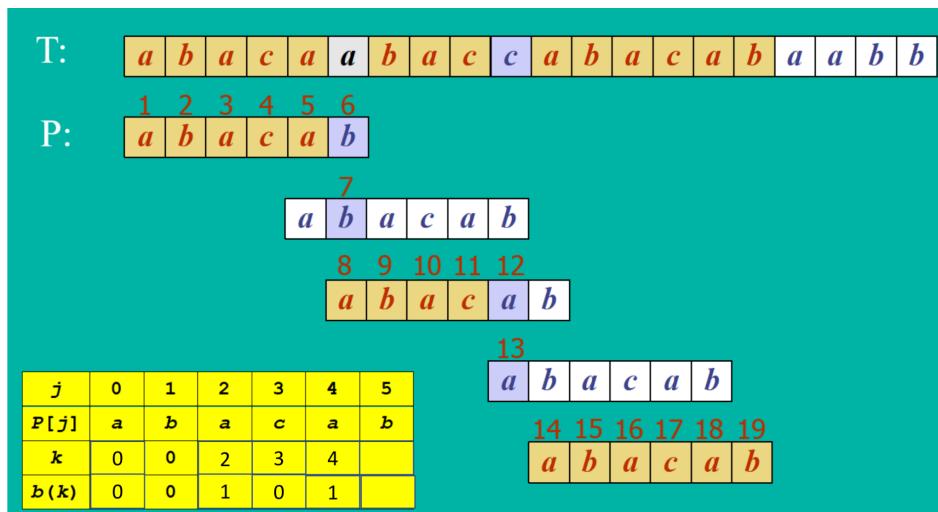
Di dalam Tugas Besar 3 ini, Anda diminta untuk mengimplementasikan sistem yang dapat melakukan identifikasi individu berbasis biometrik dengan menggunakan sidik jari. Metode yang akan digunakan untuk melakukan deteksi sidik jari adalah Boyer-Moore dan Knuth-Morris-Pratt. Selain itu, sistem ini akan dihubungkan dengan identitas sebuah individu melalui basis data sehingga harapannya terbentuk sebuah sistem yang dapat mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari.

BAB II

LANDASAN TEORI

2.1 Algoritma KMP (Knuth-Morris-Pratt)

Algoritma KMP adalah algoritma pencocokan string yang efisien untuk mencari kecocokan pola dalam teks dengan waktu kompleksitas $O(n + m)$, di mana n adalah panjang teks dan m adalah panjang pola yang dicari. Algoritma ini bekerja dengan mencocokkan pola secara iteratif untuk menghindari memeriksa kembali karakter yang telah cocok, dan ketika ada ketidakcocokan, ia menggunakan informasi yang telah diperoleh dari pencocokan sebelumnya untuk mempercepat pencarian berikutnya.



Gambar 2.1 Contoh Penggunaan Algoritma KMP

Kelebihan Algoritma KMP adalah tidak perlu mengulang mundur dalam teks input, T, membuatnya sangat cocok untuk memproses file yang sangat besar yang dibaca dari perangkat. Namun algoritma ini kurang efektif saat ukuran alfabet semakin besar karena kemungkinan ketidakcocokan yang lebih banyak. Ketidakcocokan cenderung terjadi di awal pola, namun algoritma KMP tetap lebih cepat jika ketidakcocokan terjadi di bagian akhir. Ekstensi Algoritma KMP mencakup fakta bahwa algoritma dasar tidak mempertimbangkan huruf dalam teks yang menyebabkan ketidakcocokan.

Berikut adalah *pseudocode* dari algoritma KMP.

```

KMP_Search(text, pattern):
    n = length(text)
    m = length(pattern)
    lps[m] = 0
    j = 0
    ComputeLPSArray(pattern, m, lps)
    i = 0

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

        if j == m:
            print("Pattern ditemukan pada index ", i - j)
            j = lps[j - 1]

        else if i < n and pattern[j] != text[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i = i + 1

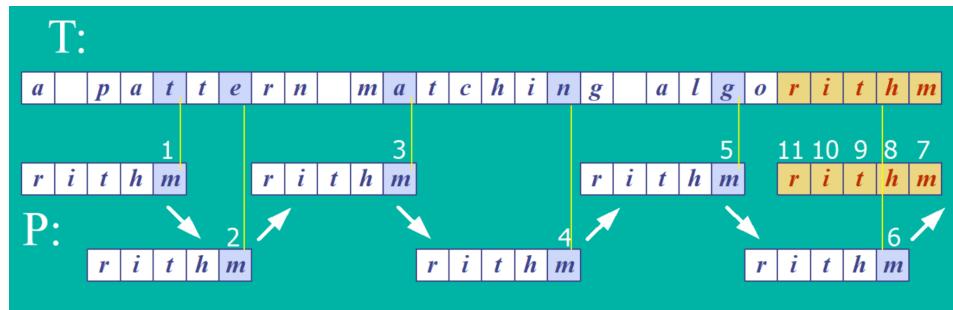
ComputeLPSArray(pattern, m, lps):
    len = 0
    lps[0] = 0
    i = 1

    while i < m:
        if pattern[i] == pattern[len]:
            len = len + 1
            lps[i] = len
            i = i + 1
        else:
            if len != 0:
                len = lps[len - 1]
            else:
                lps[i] = 0
                i = i + 1

```

2.2 Algoritma BM (Boyer-Moore)

Algoritma BM adalah algoritma pencocokan string yang juga efisien, yang bekerja dengan cara memindai teks dari kanan ke kiri. Algoritma ini memanfaatkan dua fungsi heuristik, yaitu tabel geser karakter dan tabel geser kata, untuk memindai teks dengan waktu rata-rata $O(n/m)$, di mana n adalah panjang teks dan m adalah panjang pola yang dicari.



Gambar 2.2 Contoh Penggunaan Algoritma BM

Waktu eksekusi terburuk pada algoritma Boyer-Moore adalah $O(nm + A)$. Namun, Boyer-Moore bekerja dengan cepat saat ukuran alfabet (A) besar, dan lambat saat alfabet kecil. Contohnya, algoritma ini cocok untuk teks bahasa Inggris, tetapi kurang efektif untuk data biner.

Berikut adalah *pseudocode* dari algoritma BM.

```

BM_Search(text, pattern):
    n = length(text)
    m = length(pattern)
    last = BuildLastTable(pattern, m)
    i = m - 1
    j = m - 1

    while i < n:
        if pattern[j] == text[i]:
            if j == 0:
                print("Pattern ditemukan pada index ", i)
                i = i + m - min(j + 1, last[text[i]])
                j = m - 1
            else:
                i = i - 1
                j = j - 1
        else:
            i = i + m - min(j + 1, last[text[i]])
            j = m - 1

BuildLastTable(pattern, m):
    last = array of size 256 // ASCII characters
    for each character:
        last[character] = m // default value

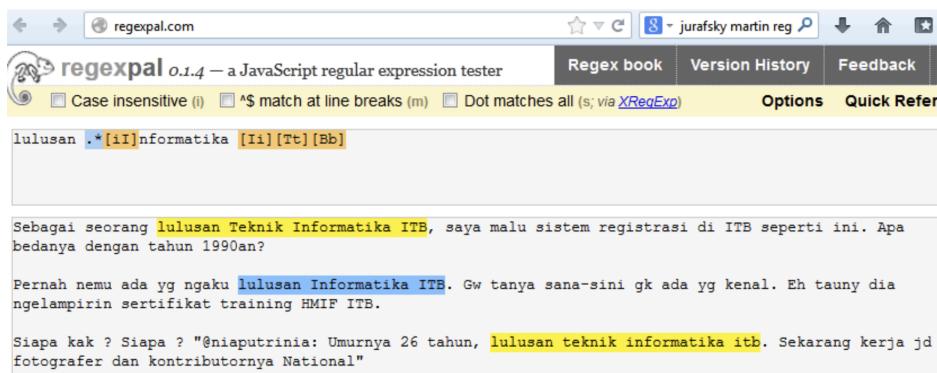
    for i from 0 to m - 2:
        last[pattern[i]] = m - 1 - i

    return last

```

2.3 Regex (Regular Expression)

Regex adalah ekspresi reguler yang merupakan urutan karakter yang membentuk pola pencarian. Regex digunakan untuk pencarian dan manipulasi teks dengan pola tertentu. Meskipun bukan algoritma pencocokan string seperti KMP atau BM, regex sangat berguna dalam mencocokkan pola teks dengan ekspresi yang lebih kompleks.



Gambar 2.3 Contoh Penggunaan Algoritma Regex

Berikut adalah *pseudocode* dari algoritma Regex.

```
Regex_Search(text, regex_pattern):
    matches = []
    for each match in text that matches regex_pattern:
        matches.append(match)
    return matches
```

2.4 Teknik Pengukuran Kemiripan Persentase

Teknik pengukuran persentase kemiripan digunakan untuk mengukur sejauh mana dua buah teks atau dokumen mirip satu sama lain. Teknik yang kami gunakan adalah Levenshtein distance. Levenshtein distance adalah ukuran jarak antara dua string. Ini didefinisikan sebagai jumlah minimum dari operasi penyisipan, penghapusan, dan penggantian yang diperlukan untuk mengubah satu string menjadi string lainnya.

Dalam program *pattern matching* ini, berikut adalah langkah-langkah detail yang diambil untuk mengukur kemiripan menggunakan Levenshtein distance:

2.4.1 Penghitungan Jarak Levenshtein

- ◆ Algoritma Levenshtein distance menggunakan matriks dua dimensi untuk melacak perhitungan jarak antara setiap karakter dari dua string.

- ◆ Baris pertama dan kolom pertama diisi dengan indeks masing-masing (0 hingga panjang string). Ini mewakili biaya penyisipan atau penghapusan karakter untuk mencapai posisi tersebut dari string kosong.
- ◆ Matriks kemudian diisi berdasarkan operasi yang diperlukan untuk mengubah substring dari satu string menjadi substring dari string lain, mempertimbangkan tiga operasi utama: penyisipan, penghapusan, dan penggantian.
- ◆ Biaya untuk setiap operasi adalah 1 kecuali karakter yang dibandingkan sama, dalam hal ini biayanya adalah 0.

2.4.2 Mengkonversi Jarak ke Kemiripan

- ◆ Setelah jarak Levenshtein dihitung, jarak tersebut diubah menjadi nilai kemiripan dengan formula:

$$\text{Kemiripan} = 1 - \frac{\text{Jarak Levenshtein}}{\text{Panjang string terpanjang}}$$

- ◆ Nilai kemiripan ini akan berada dalam rentang 0 hingga 1, di mana 1 berarti kedua string tersebut identik dan 0 berarti tidak ada kemiripan sama sekali.

2.4.3 Penggunaan Batasan Minimum Kemiripan

- ◆ Program menetapkan batasan minimum kemiripan (misalnya, 70%). Hanya nama yang memiliki nilai kemiripan lebih tinggi dari batasan ini yang akan dianggap sebagai nama yang cocok.
- ◆ Jika nilai kemiripan tertinggi di bawah batasan minimum, program akan menginformasikan bahwa tidak ada nama yang cukup mirip.

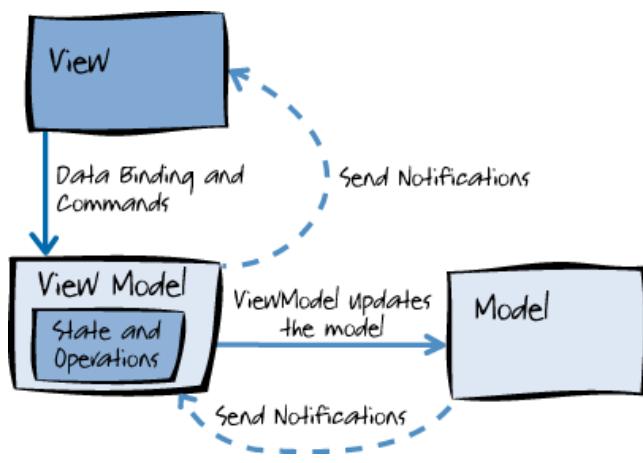
2.4.4. Implementasi dalam Kode

- ◆ Fungsi `CalculateSimilarity` mengukur kemiripan antara dua string dengan menghitung jarak Levenshtein dan kemudian mengkonversinya menjadi nilai kemiripan.

- ◆ Fungsi `FindBestMatch` menggunakan `CalculateSimilarity` untuk membandingkan nama hasil konversi dengan setiap nama dalam daftar dan menemukan nama yang memiliki kemiripan tertinggi.

2.5 Pembangunan Aplikasi Desktop

Pada Tugas Besar III Strategi Algoritma ini, Aplikasi desktop dikembangkan dengan menggunakan template proyek Avalonia di C#. Aplikasi ini dirancang untuk melakukan pencocokan data sidik jari dengan pengukuran persentase kemiripan teks ascii yang *di-convert* dari gambar sidik jari. Dengan menggunakan algoritma Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), dan Regular Expressions (Regex), aplikasi ini mampu melakukan pencarian pola dalam teks secara efisien dan akurat. Dalam pengembangannya, aplikasi ini memanfaatkan Avalonia, sebuah *framework* UI cross-platform untuk .NET. Avalonia memungkinkan aplikasi ini berjalan pada berbagai sistem operasi seperti Windows, macOS, dan Linux dengan tampilan antarmuka pengguna yang modern dan responsif. Hal ini memastikan bahwa aplikasi dapat diakses oleh pengguna dari berbagai platform tanpa kehilangan fungsi atau estetika.



Gambar 2.5 Model-View View Model

Aplikasi ini dibangun dengan mengikuti pola desain Model-View-ViewModel (MVVM) di Avalonia. Pola desain ini membantu dalam memisahkan logika *backend* dari tampilan, meningkatkan keterbacaan dan kemudahan pemeliharaan kode sehingga pengembangan aplikasi menjadi lebih modular. Fitur utama dari aplikasi ini meliputi kemampuan untuk melakukan identifikasi individu berbasis biometrik

menggunakan sidik jari, serta menghubungkan identitas individu tersebut melalui basis data. Metode identifikasi menggunakan algoritma Boyer-Moore dan Knuth-Morris-Pratt memastikan proses pencocokan sidik jari yang cepat dan akurat. Dengan demikian, aplikasi ini diharapkan dapat menjadi solusi yang aman, handal, dan mudah digunakan dalam berbagai bidang seperti kontrol akses, absensi karyawan, dan verifikasi identitas dalam transaksi keuangan.

BAB III

ANALISIS PEMECAHAN MASALAH

3.1 Langkah-Langkah Pemecahan Masalah

Permasalahan utama dari tugas besar ini adalah untuk melakukan identifikasi biometrik berbasis sidik jari. Proses identifikasi biometrik berbasis sidik jari dilakukan dengan tahapan sebagai berikut.

3.1.1 Menentukan Citra Sidik Jari

Langkah pertama dalam proses identifikasi biometrik adalah memilih citra sidik jari yang akan digunakan sebagai input. Citra ini bisa didapatkan melalui pemindaian sidik jari menggunakan perangkat keras khusus seperti fingerprint scanner atau dari basis data yang sudah ada. Citra yang diperoleh harus memiliki kualitas yang baik agar proses identifikasi berjalan dengan optimal.

3.1.2 Menentukan Area yang Cocok

Pada tahap ini akan ditentukan area tertentu pada gambar sidik jari yang akan digunakan sebagai pola (*pattern*) untuk pencocokan. Area yang dipilih pada adalah dengan melakukan segmentasi citra untuk memisahkan area yang memiliki informasi latar belakang. Setelah itu, diambil bagian tengah dari citra sidik jari yang akan dijadikan sebagai *pattern*.

3.1.3 Konversi Citra Sidik Jari ke *Binary*

Setelah area yang cocok ditentukan, citra sidik jari tersebut dikonversi ke format *binary string*. Proses ini melibatkan *thresholding*, yaitu menentukan ambang batas (*threshold*) untuk mengubah piksel citra menjadi 0 (hitam) atau 1 (putih), hal ini dilakukan untuk mengantisipasi masukan citra yang bukan hitam-putih.

3.1.4 Konversi Binari ke ASCII 8 Bits

Setelah memiliki citra binary, langkah berikutnya adalah mengkonversi *string binary* tersebut ke format ASCII 8 bits. ASCII adalah kode standar untuk pengaturan karakter yang terdiri dari 8 bits, memungkinkan setiap karakter direpresentasikan dalam bentuk biner. Konversi ini memungkinkan representasi citra dalam bentuk teks yang lebih mudah diproses oleh algoritma pencocokan.

3.1.5 Pencocokan Sidik Jari

Untuk proses pencocokan sidik jari, digunakan dua algoritma pencocokan string yaitu Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Output dari kedua algoritma ini adalah index tempat ditemukannya suatu pattern di dalam citra gambar. Jika terdapat index yang dikembalikan maka *pattern* suatu sidik jari ditemukan di *database* dengan hasil similaritas 100%.

Jika tidak terdapat index yang dikembalikan maka *pattern* suatu sidik jari tidak ditemukan di *database* dan akan ditentukan sidik jari paling mirip dengan kesamaan di atas ambang batas (*threshold*) 70%. Hal ini karena dalam praktiknya, tidak selalu ditemukan kecocokan yang sempurna untuk setiap sidik jari yang diuji dan nilai 70% dapat diperkirakan memiliki kecocokan yang pas untuk pengenalan sebuah citra, khususnya citra sidik jari. Metode perhitungan kemiripan tersebut dilakukan dengan algoritma Levenshtein Distance, yang mengukur perbedaan antara dua string berdasarkan jumlah perubahan yang diperlukan untuk mengubah satu string menjadi string lainnya. Kemudian, nilai kemiripan dihitung sebagai persentase, di mana nilai 1.0 menunjukkan string yang identik dan nilai lebih rendah menunjukkan lebih banyak perbedaan. Hal ini akan membantu menentukan kemiripan *pattern* dengan teks ASCII citra sidik jari di *database*.

3.1.6 Pencarian identitas pemilik sidik jari

Pada tahap pencarian identitas pemilik sidik jari, digunakan Regular Expression (Regex) untuk mengonversi pola karakter alay kembali ke bentuk alfabetik yang sesuai. Setelah proses konversi, dilakukan pencocokan pola antara nama yang sudah dibersihkan dengan nama yang tersimpan dalam basis data menggunakan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM).

Setelah nama yang sesuai ditemukan, sistem mengakses basis data untuk mengembalikan detail lengkap biodata individu tersebut. Dengan demikian, meskipun data nama mengalami korupsi, sistem tetap dapat melakukan identifikasi dengan memanfaatkan sidik jari dan proses pemulihan data nama yang telah dikoreksi.

3.2 Proses penyelesaian solusi dengan algoritma KMP dan BM.

3.2.1 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma KMP

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang efisien untuk menemukan kemunculan suatu pattern (pola) dalam sebuah teks. Algoritma ini menggunakan informasi dari pattern itu sendiri untuk menghindari perbandingan berulang dengan karakter yang sudah cocok. Berikut adalah proses pemetaan masalah pencocokan sidik jari menjadi elemen-elemen algoritma KMP:

1. Preprocessing (Pembentukan LPS Array)

- **Pattern (P):** Pattern sidik jari yang ingin dicari dalam citra sidik jari.
- **Longest Prefix Suffix (LPS) Array:** Array yang menyimpan panjang prefix yang juga merupakan suffix untuk setiap posisi dalam pattern. Hal ini digunakan untuk melompat dalam pattern ketika terjadi ketidakcocokan.

2. Searching

- **Text (T):** Citra sidik jari yang menjadi tempat pencarian pattern.
- **Pointer untuk Text (i):** Pointer yang berjalan melalui teks.
- **Pointer untuk Pattern (j):** Pointer yang berjalan melalui pattern.
- **Pencocokan:** Algoritma membandingkan karakter pattern dengan karakter teks. Jika karakter cocok, kedua pointer maju. Jika tidak, pointer pattern bergerak sesuai nilai dalam LPS array untuk melanjutkan pencarian tanpa perlu mengulang dari awal.

3.2.2 Proses Pemetaan Masalah menjadi Elemen-Elemen Algoritma BM

Algoritma Boyer-Moore (BM) adalah algoritma pencocokan string yang efisien dan seringkali lebih cepat daripada KMP karena menggunakan informasi dari teks serta pattern untuk melompat lebih jauh dalam proses pencarian. Berikut adalah proses pemetaan masalah pencocokan sidik jari menjadi elemen-elemen algoritma BM:

1. Preprocessing (Pembentukan Bad Character Rule dan Good Suffix Rule)

- **Pattern (P):** Pattern sidik jari yang ingin dicari dalam citra sidik jari.
- **Last Occurrence:** Tabel yang menyimpan informasi posisi terakhir kemunculan setiap karakter dalam pattern. Digunakan untuk melompat lebih jauh jika terjadi ketidakcocokan.

2. Searching

- **Text (T):** Citra sidik jari yang menjadi tempat pencarian pattern.
- **Pointer untuk Pattern (m):** Pointer yang menunjukkan posisi pattern relatif terhadap teks.
- **Pointer untuk Teks (n):** Pointer yang berjalan melalui teks.
- **Pencocokan:** Algoritma mulai membandingkan dari karakter terakhir pattern dengan teks. Jika terjadi ketidakcocokan, algoritma menggunakan Bad Character Rule atau Good Suffix Rule untuk melompat ke posisi berikutnya yang memungkinkan.

3.3 Fitur Fungsional dan Arsitektur Aplikasi Desktop yang Dibangun

Aplikasi desktop yang kami bangun menggunakan ReactiveUI dan Avalonia untuk pembuatan antarmuka pengguna dan manajemen reaktivitas. Aplikasi ini berfokus pada pemrosesan dan pencarian gambar berkas citra sidik jari. Berikut adalah penjelasan rinci tentang fitur fungsional dan arsitektur aplikasi ini:

3.3.1 Fitur Fungsional

Fitur-fitur fungsional dalam aplikasi desktop ini meliputi:

- **Pemilihan Gambar (Pilih Citra)**

Pengguna dapat memilih gambar sidik jari dari penyimpanan lokal melalui dialog pemilihan file. Gambar yang dipilih kemudian ditampilkan di aplikasi. Implementasinya menggunakan `OpenFileDialog` untuk membuka file gambar dengan ekstensi yang didukung (png, jpg, jpeg, bmp). Gambar dipilih dan ditampilkan menggunakan properti `SelectedImage` di `BioPrintViewModel`.

- **Pencarian dan Pencocokan Sidik Jari (Search)**

Pengguna dapat melakukan pencarian dan pencocokan sidik jari yang telah dipilih dengan data sidik jari yang ada. Hasil pencarian mencakup gambar hasil pencocokan, biodata pemilik sidik jari, waktu pencarian, dan persentase kecocokan. Implementasinya menggunakan `SearchCommand` di `BioPrintViewModel`. Gambar yang dipilih dikonversi dan diproses untuk pencocokan sidik jari. Hasil pencocokan ditampilkan melalui properti `SearchResultImage`, `SearchResultBioData`, `SearchTime`, dan `MatchPercentage`.

- **Navigasi Antar Halaman**

Pengguna dapat berpindah antara halaman utama dan halaman pencocokan sidik jari. Implementasinya menggunakan `BioPrintCommand` dan `StartCommand` di `MainWindowViewModel` untuk navigasi antar halaman.

- **Toggle Switch untuk Algoritma Pencocokan**

Pengguna dapat memilih metode pencocokan sidik jari menggunakan toggle switch (BM atau KMP). Implementasinya menggunakan properti `IsBMCchecked` dan `IsKMPchecked` di `BioPrintViewModel` untuk menentukan metode yang dipilih.

3.3.2 Arsitektur Aplikasi Desktop

A. ViewModels

Pada ViewModels yang digunakan untuk pembangunan aplikasi ini, kami membaginya menjadi dua file yaitu `BioPrintViewModel` dan `MainWindowViewModel`. File `BioPrintViewModel` digunakan untuk memproses logika dan data untuk halaman pencocokan sidik jari. Mengimplementasikan perintah untuk pemilihan gambar, pencarian, dan penyimpanan gambar yang dipilih. Sedangkan file `MainWindowViewModel` digunakan untuk memproses logika dan data untuk halaman utama aplikasi. Mengimplementasikan perintah untuk navigasi ke halaman pencocokan sidik jari.

B. Views

Arsitektur views yang kami gunakan dalam pembangunan aplikasi ini disesuaikan dengan viewmodels yang telah disebutkan sebelumnya sehingga terdapat dua view yaitu BioPrint dan MainWindow. View BioPrint merupakan tampilan untuk halaman pencocokan sidik jari. Menggunakan XAML untuk mendefinisikan antarmuka pengguna, termasuk tombol, gambar, toggle switch, dan teks hasil pencarian. Sedangkan MainWindow merupakan tampilan untuk halaman utama. Menggunakan XAML untuk mendefinisikan antarmuka pengguna, termasuk tombol mulai.

C. Model

Pada arsitektur model aplikasi program yang kami bangun terdapat kelas FingerprintConverter dan Processing. Kelas tersebut digunakan untuk mengkonversi gambar dan memproses pencocokan sidik jari. Menerima input gambar dan metode pencocokan, kemudian menghasilkan hasil pencarian yang mencakup biodata, waktu pencarian, dan persentase kecocokan.

D. Reactive Commands

ReactiveCommand digunakan untuk mengimplementasikan perintah yang dapat dipanggil dari UI, seperti PilihCitraCommand, SearchCommand, dan BioPrintCommand. ReactiveCommand memungkinkan eksekusi perintah secara reaktif dan mengikatnya ke UI.

E. Binding dan DataContext

Data binding digunakan secara luas untuk menghubungkan ViewModels dengan Views. Properti-properti di ViewModels terikat ke elemen-elemen UI di Views menggunakan binding. DataContext dari setiap View diatur ke instance ViewModel yang sesuai.

F. Asynchronous Operations

Operasi yang memerlukan waktu, seperti pembukaan file dan pencocokan sidik jari, dilakukan secara asinkron menggunakan Task.Run untuk menjaga responsivitas UI.

3.4 Ilustrasi Kasus



Misalkan akan dilakukan identifikasi sebuah citra sidik jari dari gambar seperti di samping. Dari citra sidik jari tersebut akan diambil sebuah area yang akan dijadikan *pattern*. *Preprocessing* dilakukan untuk menghilangkan tepian informasi latar belakang dari citra sidik jari. Setelah itu, akan dipilih sebesar 64 karakter di tengah-tengah citra sidik jari.

Ilustrasi pengambilannya dapat dimisalkan sebagai berikut. Potongan konversi binary dan ASCII bukan yang sebenarnya, hanya sebagai ilustrasi saja.

Potongan Konversi Binary												Potongan ASCII 8-Bits (<i>Pattern</i>)																																																																																																																																			
<table border="1"> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>...</td><td>...</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr> </table>												1	0	1	1	1	0	0	0	0	1	1	0	0	1	0	1	1	1	0	0	0	1	1	0	1	1	1	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	1	1	1	0	1	1	1	0	0	1	0	0	0	1	0	0	1	1	A B A C A B											
...																																																																																																																																				
...	1	0	1	1	1	0	0	0	0																																																																																																																																				
...	1	1	0	0	1	0	1	1	1																																																																																																																																				
...	0	0	0	1	1	0	1	1	1																																																																																																																																				
...	1	0	0	1	0	1	1	0																																																																																																																																				
...	0	1	1	1	0	0	1	0																																																																																																																																				
...	1	0	0	1	0	0	1	1																																																																																																																																				
...	1	0	1	1	1	0	0	1																																																																																																																																				
...	0	0	0	1	0	0	1	1																																																																																																																																				
...																																																																																																																																				

Misalkan *pattern* dari citra sidik jari yang ingin dicocokkan diwakili oleh string sebagai berikut.

- Pattern (P): "A B A C A B"

Pattern tersebut akan dilakukan pencocokan string terhadap citra sidik jari yang terdapat pada *database*. Citra sidik jari di *database* akan dilakukan konversi secara keseluruhan menjadi ASCII 8-Bits sebagai teks tempat mencocokkan *pattern*.

Pencocokan *pattern* dengan teks dilakukan dengan menggunakan dua algoritma, yaitu Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM).

1. Knuth-Morris-Pratt (KMP)

Ketika menggunakan algoritma KMP, terdapat *preprocessing* yang harus dilakukan yaitu untuk menghitung *border function*. Hal ini dilakukan dengan membangun LPS (*Longest Prefix Suffix*) array untuk *pattern*. Misalkan pada pencocokan ke-*n*, terdapat citra sidik jari pada database dengan hasil konversi ke ASCII 8-Bits sebagai berikut.

Teks (T): "A B A C A A B A C C A B A C A B A A B"

Berikut LPS yang diperoleh dari ilustrasi tersebut.

j	0	1	2	3	4	5
$P[j]$	A	B	A	C	A	B
k	-	0	1	2	3	4
$b[k]$	-	0	0	1	0	1

Proses pencocokan dengan algoritma KMP dilakukan sebagai berikut.

T	A	B	A	C	A	A	B	A	C	C	A	B	A	C	A	B	A	A	B
P	A	B	A	C	A	B													
1	2	3	4	5	6														
						A	B	7	A	C	A	B							
						A	B	8	A	C	A	B							
									A	B	A	C	A	B					
										A	B	A	C	A	B				
											A	B	A	C	A	B			
												A	B	A	C	A	B		
													A	B	A	C	A	B	
														A	B	A	C	A	B

Dengan menggunakan algoritma KMP diperoleh *pattern* ditemukan pada indeks teks ke-10, dengan jumlah perbandingan 19 kali.

2. Boyer-Moore (BM)

Ketika menggunakan algoritma BM, terdapat *preprocessing* yang harus dilakukan yaitu untuk menghitung *last occurrences* tiap karakter yang ada di teks. Hal ini dilakukan dengan membangun *last occurrence array*.

Berikut *last occurrence* array yang diperoleh dari ilustrasi tersebut. Misalkan pada pencocokan ke- n , terdapat citra sidik jari pada database dengan hasil konversi ke ASCII 8-Bits sebagai berikut.

Teks (T): "A B A C A A B A D C A B A C A B A A B"

x	A	B	C	lainnya
$L[x]$	4	5	3	-1

Proses pencocokan dengan algoritma BM dilakukan sebagai berikut.

T	A	B	A	C	A	A	B	A	D	C	A	B	A	C	A	B	A	A	B
P	A	B	A	C	A	B 1													
		A	B	A	C 4	A 3	B 2												
		A	B	A	C	A	B 5												
			A	B	A	C	A	B 6											
									A	B	A	C	A	B 7					
										A 13	B 12	A 11	C 10	A 9	B 8				

Dengan menggunakan algoritma KMP diperoleh *pattern* ditemukan pada indeks teks ke-10, dengan jumlah perbandingan 13 kali.

Jika tidak terdapat index yang dikembalikan maka *pattern* suatu sidik jari tidak ditemukan di *database* dan akan ditentukan sidik jari paling mirip dengan kesamaan di atas ambang batas (*threshold*) 70%. Hal ini karena dalam praktiknya, tidak selalu ditemukan kecocokan yang sempurna untuk setiap sidik jari yang diuji dan nilai 70% dapat diperkirakan memiliki kecocokan yang pas untuk pengenalan sebuah citra, khususnya citra sidik jari. Metode perhitungan kemiripan tersebut dilakukan dengan algoritma Levenshtein Distance, yang mengukur perbedaan antara dua string berdasarkan jumlah perubahan yang diperlukan untuk mengubah satu string menjadi string lainnya. Kemudian, nilai kemiripan dihitung sebagai persentase, di mana nilai 1.0 menunjukkan string yang identik dan nilai lebih rendah menunjukkan lebih banyak perbedaan. Hal ini akan membantu menentukan kemiripan *pattern* dengan teks ASCII citra sidik jari di *database*.

Pada tahap pencarian identitas pemilik sidik jari, digunakan Regular Expression (Regex) untuk mengonversi pola karakter alay kembali ke bentuk alfabetik yang sesuai. Setelah proses konversi, dilakukan pencocokan pola antara nama yang sudah dibersihkan dengan nama yang tersimpan dalam basis data menggunakan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM).

Setelah nama yang sesuai ditemukan, sistem mengakses basis data untuk mengembalikan detail lengkap biodata individu tersebut. Dengan demikian, meskipun data nama mengalami korupsi, sistem tetap dapat melakukan identifikasi dengan memanfaatkan sidik jari dan proses pemulihan data nama yang telah dikoreksi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

Berikut merupakan *source code* algoritma utama yang memuat logika utama pada program.

```
BMAlgorithm.cs

using System;
using System.Collections.Generic;

namespace HapHipHop.Models
{
    public static class BMAlgorithm
    {
        public static (List<int> positions, int comparisons, List<(int, int, char, char)> comparedCharacters) BoyerMooreSearch(string pat, string txt)
        {
            int m = pat.Length;
            int n = txt.Length;

            if (m > n)
            {
                return (new List<int>(), 0, new List<(int, int, char, char)>());
            }

            List<int> results = new List<int>();
            int comparisons = 0;
            List<(int, int, char, char)> comparedCharacters = new List<(int, int, char, char)>();

            int[] badChar = new int[256];
            BadCharHeuristic(pat, m, badChar);

            int s = 0;
            while (s <= (n - m))
            {
                int j = m - 1;

                while (j >= 0 && pat[j] == txt[s + j])
```

```
{  
    comparisons++;  
    comparedCharacters.Add((s + j, j, txt[s + j], pat[j]));  
    j--;  
}  
  
if (j < 0)  
{  
    results.Add(s);  
    s += (s + m < n) ? m - badChar[txt[s + m]] : 1;  
}  
else  
{  
    comparisons++;  
    comparedCharacters.Add((s + j, j, txt[s + j], pat[j]));  
    s += Math.Max(1, j - badChar[txt[s + j]]);  
}  
}  
  
return (results, comparisons, comparedCharacters);  
}  
  
private static void BadCharHeuristic(string str, int size, int[]  
badChar)  
{  
    for (int i = 0; i < 256; i++)  
        badChar[i] = -1;  
  
    for (int i = 0; i < size; i++)  
        badChar[(int)str[i]] = i;  
}
```

KMPAlgorithm.cs

```
using System;  
using System.Collections.Generic;  
  
namespace HapHipHop.Models  
{  
    public static class KMPAlgorithm
```

```
{  
    public static (List<int> positions, int comparisons, List<(int,  
    int, char, char)> comparedCharacters) KMPSearch(string pat, string txt)  
    {  
        int M = pat.Length;  
        int N = txt.Length;  
  
        int[] lps = new int[M];  
        int j = 0;  
        int comparisons = 0;  
        List<(int, int, char, char)> comparedCharacters = new  
        List<(int, int, char, char)>();  
  
        ComputeLPSArray(pat, M, lps);  
  
        int i = 0;  
        List<int> results = new List<int>();  
        while (i < N)  
        {  
            comparisons++;  
            comparedCharacters.Add((i, j, txt[i], pat[j]));  
            if (pat[j] == txt[i])  
            {  
                j++;  
                i++;  
            }  
  
            if (j == M)  
            {  
                results.Add(i - j);  
                j = lps[j - 1];  
            }  
            else if (i < N && pat[j] != txt[i])  
            {  
                if (j != 0)  
                {  
                    j = lps[j - 1];  
                }  
                else  
                {  
                    i++;  
                }  
            }  
        }  
    }  
}
```

```
        }

    }

    return (results, comparisons, comparedCharacters);
}

private static void ComputeLPSArray(string pat, int M, int[] lps)
{
    int len = 0;
    int i = 1;
    lps[0] = 0;

    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len != 0)
            {
                len = lps[len - 1];
            }
            else
            {
                lps[i] = len;
                i++;
            }
        }
    }
}
```

ImageConverter.cs

```
using System;
using System.IO;
using System.Text;
using System.Runtime.InteropServices;
```

```
using Avalonia;
using Avalonia.Media.Imaging;
using Avalonia.Platform;

namespace HapHipHop.Models
{
    public static class FingerprintConverter
    {
        public static string ConvertImageToBinary(Bitmap image)
        {
            StringBuilder binaryStringBuilder = new StringBuilder();

            int width = image.PixelSize.Width;
            int height = image.PixelSize.Height;
            int stride = width * 4;

            byte[] pixelData = new byte[height * stride];
            var pixelRect = new PixelRect(0, 0, width, height);
            var pixelFormat = PixelFormat.Bgra8888;

            GCHandle handle = GCHandle.Alloc(pixelData,
GCHandleType.Pinned);
            try
            {
                IntPtr pointer = handle.AddrOfPinnedObject();
                image.CopyPixels(pixelRect, (nint)pointer,
pixelData.Length, stride);
            }
            finally
            {
                handle.Free();
            }

            for (int y = 0; y < height; y++)
            {
                for (int x = 0; x < width; x++)
                {
                    int index = y * stride + x * 4;
                    byte b = pixelData[index];
                    byte g = pixelData[index + 1];
                    byte r = pixelData[index + 2];
                    binaryStringBuilder.Append(b);
                    binaryStringBuilder.Append(g);
                    binaryStringBuilder.Append(r);
                }
            }
        }
    }
}
```

```
        int binaryValue = (r == 0 && g == 0 && b == 0) ? 0 :  
1;  
        binaryStringBuilder.Append(binaryValue);  
    }  
}  
  
return binaryStringBuilder.ToString();  
}  
  
public static string ConvertBinaryToAscii(string binaryString)  
{  
    StringBuilder asciiStringBuilder = new StringBuilder();  
  
    for (int i = 0; i < binaryString.Length; i += 8)  
    {  
        if (i + 8 <= binaryString.Length)  
        {  
            string byteString = binaryString.Substring(i, 8);  
            byte byteValue = Convert.ToByte(byteString, 2);  
            char asciiChar = (char)byteValue;  
            asciiStringBuilder.Append(asciiChar);  
        }  
    }  
  
    return asciiStringBuilder.ToString();  
}  
  
public static string CleanPattern(string source, string pattern)  
{  
    source = RemoveFirstLongestOccurrence(source, pattern);  
    source = RemoveLastLongestOccurrence(source, pattern);  
  
    int patternSize = source.Length;  
    if (patternSize <= 64)  
    {  
        return source;  
    }  
    else  
    {  
        int middleIdx = patternSize / 2;  
        int startIdx = middleIdx - 32;  
        if (startIdx < 0)
```

```
{  
    startIdx = 0;  
}  
else if (startIdx + 64 > patternSize)  
{  
    startIdx = patternSize - 64;  
}  
  
return source.Substring(startIdx, 64);  
}  
}  
  
public static string RemoveFirstLongestOccurrence(string source,  
string pattern)  
{  
    int longestSequenceLength = 0;  
    int i = 0;  
  
    while (i <= source.Length - pattern.Length)  
    {  
        if (source.Substring(i, pattern.Length) == pattern)  
        {  
            longestSequenceLength += pattern.Length;  
            i += pattern.Length;  
        }  
        else  
        {  
            break;  
        }  
    }  
  
    source = source.Remove(0, longestSequenceLength);  
    return source;  
}  
  
public static string RemoveLastLongestOccurrence(string source,  
string pattern)  
{  
    int startIndex = source.Length;  
    int i = source.Length;  
    while (i >= pattern.Length)  
    {  
        if (source.Substring(i - pattern.Length, pattern.Length) == pattern)  
        {  
            startIndex = i - pattern.Length;  
        }  
        i -= pattern.Length;  
    }  
  
    source = source.Remove(startIndex, source.Length - startIndex);  
    return source;  
}
```

```
        if (source.Substring(i - pattern.Length, pattern.Length)
== pattern)
{
    startIndex = i - pattern.Length;
    i -= pattern.Length;
}
else
{
    break;
}
}

source = source.Remove(startIndex);
return source;
}
}
}
```

Processing.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Npgsql;
using Avalonia.Media.Imaging;

namespace HapHipHop.Models
{
    public class Processing
    {
        public class Biodata
        {
            public string NIK { get; set; }
            public string Nama { get; set; }
            public string TempatLahir { get; set; }
            public DateTime TanggalLahir { get; set; }
            public string JenisKelamin { get; set; }
            public string GolonganDarah { get; set; }
            public string Alamat { get; set; }
        }
    }
}
```

```
public string Agama { get; set; }
public string StatusPerkawinan { get; set; }
public string Pekerjaan { get; set; }
public string Kewarganegaraan { get; set; }

public override string ToString()
{
    return $"NIK: {NIK}\nNama: {Nama}\nTempat Lahir:
{TempatLahir}\nTanggal Lahir: {TanggalLahir:yyyy-MM-dd}\n" +
        $"Jenis Kelamin: {JenisKelamin}\nGolongan Darah:
{GolonganDarah}\nAlamat: {Alamat}\nAgama: {Agama}\n" +
        $"Status Perkawinan: {StatusPerkawinan}\nPekerjaan:
{Pekerjaan}\nKewarganegaraan: {Kewarganegaraan}";
}

public static (string bestPath, Biodata biodata, double time,
double percentage) ProcessFingerprintMatching(Bitmap inputImage, bool
algorithmChoice)
{
    var basePath = AppDomain.CurrentDomain.BaseDirectory;
    var relativePath = Path.Combine(basePath, "..", "..", "test");
    var absolutePath = Path.GetFullPath(relativePath);

    if (!Directory.Exists(absolutePath))
    {
        Directory.CreateDirectory(absolutePath);
    }

    string logFilePath = Path.Combine(absolutePath, "log.txt");
    string notFoundPath = Path.Combine(absolutePath,
"notfound.txt");

    try
    {
        var imageFilesTask = Task.Run(() =>
LoadImageFilesFromDatabaseAsync());
        var biodataListTask = Task.Run(() => LoadBiodataAsync());

        Task.WaitAll(imageFilesTask, biodataListTask);

        var imageFiles = imageFilesTask.Result;
```

```
var biodataList = biodataListTask.Result;

    if (inputImage == null)
    {
        return (string.Empty, new Biodata(), 0, 0);
    }

    string inputPattern =
FingerprintConverter.ConvertImageToBinary(inputImage);
    inputPattern =
FingerprintConverter.ConvertBinaryToAscii(inputPattern);

    string pattern =
FingerprintConverter.CleanPattern(inputPattern, "ÿÿÿÿÿÿÿÿÿÿ");
    pattern = FingerprintConverter.CleanPattern(inputPattern,
"ÿ");
        // File.AppendAllText(Path.CombineAbsolutePath,
"pattern.txt"), pattern);
        // File.AppendAllText(Path.CombineAbsolutePath,
"inputpattern.txt"), inputPattern);
    int algorithm = algorithmChoice ? 2 : 1;

    string bestMatchOwner = "";
    double highestSimilarity = 0;
    string bestPathImage = "";

Stopwatch stopwatch = Stopwatch.StartNew();

var tasks = imageFiles.AsParallel().Select(imageFile =>
{
    try
    {
        if (!File.Exists(imageFile.imagePath))
        {
            File.AppendAllText(notFoundPath, $"File not
found: {imageFile.imagePath}\n");
            return (similarity: 0.0, ownerName:
imageFile.ownerName, imagePath: imageFile.imagePath);
        }
    }

    using (Bitmap dbImage = new
Bitmap(imageFile.imagePath))
```

```
{  
    string dbText =  
FingerprintConverter.ConvertImageToBinary(dbImage);  
    dbText =  
FingerprintConverter.ConvertBinaryToAscii(dbText);  
  
    double similarity = 0;  
    if (algorithm == 1)  
    {  
        var result =  
KMPAlgorithm.KMPSearch(pattern, dbText);  
        similarity = result.positions.Count > 0 ?  
1.0 : RegexString.CalculateSimilarity(pattern, dbText);  
    }  
    else if (algorithm == 2)  
    {  
        var result =  
BMAlgorithm.BoyerMooreSearch(pattern, dbText);  
        similarity = result.positions.Count > 0 ?  
1.0 : RegexString.CalculateSimilarity(pattern, dbText);  
    }  
  
    File.AppendAllText(logFilePath, $"Similarity:  
{similarity:P2} Owner: {imageFile.ownerName} Path:  
{imageFile.imagePath}\n");  
    return (similarity, imageFile.ownerName,  
imageFile.imagePath);  
}  
}  
catch (Exception)  
{  
    return (similarity: 0.0, ownerName:  
imageFile.ownerName, imagePath: imageFile.imagePath);  
}  
}).ToArray();  
  
foreach (var task in tasks)  
{  
    var result = task;  
    if (result.similarity > highestSimilarity)  
    {  
        highestSimilarity = result.similarity;  
    }  
}
```

```
        bestMatchOwner = result.ownerName;
        bestPathImage = result.imagePath;
    }
}

List<string> namaAlay = biodataList.Select(biodata =>
RegexString.ConvertAlayToOriginal(biodata>Nama)).ToList();

double similarityThreshold = 0.7;

string bestMatch =
RegexString.FindBestMatch(bestMatchOwner, namaAlay, similarityThreshold,
out double similarity);

stopwatch.Stop();

string bestPath = bestPathImage;
Biodata bestBiodata = new Biodata();
double time = stopwatch.ElapsedMilliseconds;
double percentage = highestSimilarity * 100;

if (highestSimilarity < similarityThreshold)
{
    return (string.Empty, new Biodata(), time,
percentage);
}
else
{
    foreach (var biodata in biodataList)
    {
        if
(RegexString.ConvertAlayToOriginal(biodata>Nama) == bestMatch)
        {
            bestBiodata = biodata;
            bestBiodata>Nama = bestMatchOwner;
            break;
        }
    }
    return (bestPath, bestBiodata, time, percentage);
}
}

catch (Exception)
```

```
{  
    return (string.Empty, new Biodata(), 0, 0);  
}  
}  
  
public static async Task<List<(string imagePath, string  
ownerName)>> LoadImageFilesFromDatabaseAsync()  
{  
    var imageFiles = new List<(string imagePath, string  
ownerName)>();  
    string connectionString =  
"Host=localhost;Username=postgres;Password=3663;Database=Tubes3_HapHipHop  
";  
    string query = "SELECT berkas_citra, nama FROM sidik_jari";  
    var basePath = AppDomain.CurrentDomain.BaseDirectory;  
    var relativePath = Path.Combine(basePath, "...", "...", "...",  
    "...", "test");  
    var absolutePath = Path.GetFullPath(relativePath);  
  
    try  
    {  
        using (var connection = new  
NpgsqlConnection(connectionString))  
        {  
            await connection.OpenAsync();  
            using (var command = new NpgsqlCommand(query,  
connection))  
            {  
                using (var reader = await  
command.ExecuteReaderAsync())  
                {  
                    while (await reader.ReadAsync())  
                    {  
                        string imagePath =  
Path.Combine(absolutePath, reader.GetString(0));  
                        string ownerName = reader.GetString(1);  
                        imagePath = imagePath.Replace("/", "\\\\";  
                        File.AppendAllText("imagepath.txt",  
imagePath + "\n");  
                        imageFiles.Add((imagePath, ownerName));  
                    }  
                }  
            }  
        }  
    }
```

```
        }

    }

}

catch (Exception ex)
{
    // exceptions
}

return imageFiles;
}

public static async Task<List<Biodata>> LoadBiodataAsync()
{
    var biodataList = new List<Biodata>();
    string connectionString =
"Host=localhost;Username=postgres;Password=3663;Database=Tubes3_HapHipHop";
    string query = "SELECT nik, nama, tempat_lahir, tanggal_lahir,
jenis_kelamin, golongan_darah, alamat, agama, status_perkawinan,
pekerjaan, kewarganegaraan FROM biodata";

    try
    {
        using (var connection = new
NpgsqlConnection(connectionString))
        {
            await connection.OpenAsync();
            using (var command = new NpgsqlCommand(query,
connection))
            {
                using (var reader = await
command.ExecuteReaderAsync())
                {
                    while (await reader.ReadAsync())
                    {
                        var biodata = new Biodata
                        {
                            NIK =
SimpleAES.Decrypt(reader.GetString(0), "abcdefghijklmnopqrstuvwxyz"),
                            Nama =
SimpleAES.Decrypt(reader.GetString(1), "abcdefghijklmnopqrstuvwxyz"),
                            TempatLahir =

```

```
SimpleAES.Decrypt(reader.GetString(2), "abcdefghijklmnopqrstuvwxyz"),
                    TanggalLahir =
SimpleAES.DecryptDate(reader.GetString(3), "abcdefghijklmnopqrstuvwxyz"),
                    JenisKelamin = reader.GetString(4),
                    GolonganDarah =
SimpleAES.Decrypt(reader.GetString(5), "abcdefghijklmnopqrstuvwxyz"),
                    Alamat =
SimpleAES.Decrypt(reader.GetString(6), "abcdefghijklmnopqrstuvwxyz"),
                    Agama =
SimpleAES.Decrypt(reader.GetString(7), "abcdefghijklmnopqrstuvwxyz"),
                    StatusPerkawinan =
reader.GetString(8),
                    Pekerjaan =
SimpleAES.Decrypt(reader.GetString(9), "abcdefghijklmnopqrstuvwxyz"),
                    Kewarganegaraan =
SimpleAES.Decrypt(reader.GetString(10), "abcdefghijklmnopqrstuvwxyz")
                    };
                    biodataList.Add(biodata);
                }
            }
        }
    }

    return biodataList;
}
}
```

RegexString.cs

```
using System;
using System.Collections.Generic;

namespace HapHipHop.Models
{
    public class RegexString
    {
        ...
    }
}
```

```
public static string ConvertAlayToOriginal(string input)
{
    var replacements = new List<(string pattern, string
replacement)>
    {
        ("1", "i"), ("1", "l"), ("2", "z"), ("3", "e"), ("4", "a"),
("5", "s"),
        ("6", "g"), ("7", "t"), ("8", "b"), ("9", "g"), ("0", "o"),
("@", "a"),
        ("!", "i"), ("$", "s"), ("&", "e"), ("#", "h")
    };

    input = ReplaceAll(input, replacements);

    input = RemoveNonAlphanumeric(input);

    var abbreviations = new List<(string pattern, string
replacement)>
    {
        ("bntng", "bintang"),
        ("dw", "dwi"),
        ("mrthn", "marthen"),
    };

    input = ReplaceAll(input, abbreviations);

    input = CapitalizeProperly(input);

    return input;
}

static string ReplaceAll(string input, List<(string pattern,
string replacement)> replacements)
{
    foreach (var (pattern, replacement) in replacements)
    {
        input = Replace(input, pattern, replacement);
    }
    return input;
}

static string Replace(string input, string pattern, string
```

```
replacement)
{
    int index = input.IndexOf(pattern,
StringComparison.OrdinalIgnoreCase);
    while (index != -1)
    {
        input = input.Substring(0, index) + replacement +
input.Substring(index + pattern.Length);
        index = input.IndexOf(pattern, index + replacement.Length,
StringComparison.OrdinalIgnoreCase);
    }
    return input;
}

static string RemoveNonAlphanumeric(string input)
{
    char[] result = new char[input.Length];
    int resultIndex = 0;

    foreach (char c in input)
    {
        if (char.IsLetter(c) || char.IsWhiteSpace(c))
        {
            result[resultIndex++] = c;
        }
    }

    return new string(result, 0, resultIndex);
}

static string CapitalizeProperly(string input)
{
    string[] words = input.Split(' ',
StringSplitOptions.RemoveEmptyEntries);
    for (int i = 0; i < words.Length; i++)
    {
        if (words[i].Length > 0)
        {
            words[i] = char.ToUpper(words[i][0]) +
words[i].Substring(1).ToLower();
        }
    }
}
```

```
        return string.Join(" ", words);
    }

    public static string FindBestMatch(string input, List<string>
names, double similarityThreshold, out double highestSimilarity)
{
    string? bestMatch = null;
    highestSimilarity = 0.0;

    foreach (var name in names)
    {
        double similarity = CalculateSimilarity(input, name);
        if (similarity > highestSimilarity)
        {
            highestSimilarity = similarity;
            bestMatch = name;
        }
    }

    return highestSimilarity >= similarityThreshold ? bestMatch ??
string.Empty : "Tidak ada kecocokan yang memadai";
}

public static double CalculateSimilarity(string source, string
target)
{
    if (string.IsNullOrEmpty(source) || string.IsNullOrEmpty(target)) return 0.0;
    if (source == target) return 1.0;

    int stepsToSame = GetMaxSimilarity(source, target);
    return (1.0 - ((double)stepsToSame / Math.Max(source.Length,
target.Length)));
}

static int GetMaxSimilarity(string source, string target)
{
    int maxSimilarity = int.MaxValue;
    int sourceLength = source.Length;
    int targetLength = target.Length;

    for (int i = 0; i <= targetLength - sourceLength; i++)
    {
        int currentSimilarity = 0;
        for (int j = 0; j < sourceLength; j++)
        {
            if (source[j] == target[i + j])
                currentSimilarity++;
        }
        if (currentSimilarity > maxSimilarity)
            maxSimilarity = currentSimilarity;
    }
    return maxSimilarity;
}
```

```
{  
    string subTarget = target.Substring(i, sourceLength);  
    int similarity = ComputeLevenshteinDistance(source,  
subTarget);  
    if (similarity < maxSimilarity)  
    {  
        maxSimilarity = similarity;  
    }  
}  
  
return maxSimilarity;  
}  
  
static int ComputeLevenshteinDistance(string source, string  
target)  
{  
    int n = source.Length;  
    int m = target.Length;  
    int[,] d = new int[n + 1, m + 1];  
  
    if (n == 0) return m;  
    if (m == 0) return n;  
  
    for (int i = 0; i <= n; d[i, 0] = i++) {}  
    for (int j = 0; j <= m; d[0, j] = j++) {}  
  
    for (int i = 1; i <= n; i++)  
    {  
        for (int j = 1; j <= m; j++)  
        {  
            int cost = (target[j - 1] == source[i - 1]) ? 0 : 1;  
            d[i, j] = Math.Min(  
                Math.Min(d[i - 1, j] + 1, d[i, j - 1] + 1),  
                d[i - 1, j - 1] + cost);  
        }  
    }  
  
    return d[n, m];  
}  
}  
}
```

Encryption.cs (Bonus)

```
using System;
using System.Text;
using System.Globalization;

namespace HapHipHop.Models
{
    public class SimpleAES
    {
        private static readonly byte[] sBox = new byte[256] {
            0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
            0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
            0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
            0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
            0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
            0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
            0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
            0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
            0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
            0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
            0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
            0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
            0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
            0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
            0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
            0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
            0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
            0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
            0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
            0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
            0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
            0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
            0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
            0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
            0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
            0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
            0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
            0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
            0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
            0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
            0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
            0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
        };
    }
}
```

```
};

private static readonly byte[] invSBox = new byte[256] {
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38,
    0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87,
    0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d,
    0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2,
    0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16,
    0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda,
    0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0xa,
    0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02,
    0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea,
    0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85,
    0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89,
    0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20,
    0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31,
    0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d,
    0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0,
    0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26,
    0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d
};

private static byte[] SubBytes(byte[] state)
{
    for (int i = 0; i < state.Length; i++)
    {
        state[i] = sBox[state[i]];
    }
}
```

```
        }

        return state;
    }

    private static byte[] InvSubBytes(byte[] state)
    {
        for (int i = 0; i < state.Length; i++)
        {
            state[i] = invSBox[state[i]];
        }
        return state;
    }

    private static byte[] AddRoundKey(byte[] state, byte[] roundKey)
    {
        for (int i = 0; i < state.Length; i++)
        {
            state[i] ^= roundKey[i];
        }
        return state;
    }

    private static byte[] PadRight(byte[] input, int length)
    {
        byte[] padded = new byte[length];
        Array.Copy(input, padded, input.Length);
        for (int i = input.Length; i < length; i++)
        {
            padded[i] = 0x00;
        }
        return padded;
    }

    public static string Encrypt(string plainText, string key)
    {
        byte[] keyBytes = Encoding.UTF8.GetBytes(key.PadRight(16));
        byte[] plainBytes = Encoding.UTF8.GetBytes(plainText);
        int numberOfBlocks = (plainBytes.Length + 15) / 16;
        byte[] cipherBytes = new byte[numberOfBlocks * 16];

        for (int blockIndex = 0; blockIndex < numberOfBlocks;
blockIndex++)
```

```
{  
    byte[] block = new byte[16];  
    int bytesToCopy = Math.Min(16, plainBytes.Length -  
blockIndex * 16);  
    Array.Copy(plainBytes, blockIndex * 16, block, 0,  
bytesToCopy);  
  
    block = AddRoundKey(block, keyBytes);  
    block = SubBytes(block);  
    block = AddRoundKey(block, keyBytes);  
  
    Array.Copy(block, 0, cipherBytes, blockIndex * 16, 16);  
}  
  
return Convert.ToBase64String(cipherBytes);  
}  
  
public static string Decrypt(string cipherText, string key)  
{  
    try  
    {  
        byte[] keyBytes =  
Encoding.UTF8.GetBytes(key.PadRight(16));  
        byte[] cipherBytes = Convert.FromBase64String(cipherText);  
        int numberOfBlocks = cipherBytes.Length / 16;  
        byte[] plainBytes = new byte[cipherBytes.Length];  
  
        for (int blockIndex = 0; blockIndex < numberOfBlocks;  
blockIndex++)  
        {  
            byte[] block = new byte[16];  
            Array.Copy(cipherBytes, blockIndex * 16, block, 0,  
16);  
  
            block = AddRoundKey(block, keyBytes);  
            block = InvSubBytes(block);  
            block = AddRoundKey(block, keyBytes);  
  
            Array.Copy(block, 0, plainBytes, blockIndex * 16, 16);  
        }  
  
        return Encoding.UTF8.GetString(plainBytes).TrimEnd('\0');  
    }
```

```
        }

        catch (FormatException ex)
        {
            Console.WriteLine("Error decoding Base64 string: " +
ex.Message);
            return null;
        }
    }

    public static string EncryptDate(DateTime date, string key)
{
    string dateString = date.ToString("yyyy-MM-dd",
CultureInfo.InvariantCulture);
    return Encrypt(dateString, key);
}

    public static DateTime DecryptDate(string cipherText, string key)
{
    string dateString = Decrypt(cipherText, key);
    return DateTime.ParseExact(dateString, "yyyy-MM-dd",
CultureInfo.InvariantCulture);
}
}
```

4.1.1 Struktur Data

Pada tugas besar ini, terdapat beberapa struktur data yang digunakan, yaitu:

1. **List<T>**, merupakan struktur data yang menyimpan elemen atau objek dalam bentuk list. Struktur data ini digunakan pada berbagai bagian program, termasuk untuk menyimpan hasil pencarian posisi karakter yang ditemukan, jumlah perbandingan karakter, dan karakter yang dibandingkan selama proses pencarian pola dalam teks menggunakan algoritma Boyer-Moore dan KMP.
2. **Array**, merupakan struktur data yang menyimpan elemen-elemen dengan tipe yang sama dalam satu variabel. Pada program ini, array digunakan untuk menyimpan nilai-nilai tabel heuristik karakter buruk dalam algoritma Boyer-Moore dan nilai-nilai tabel lps (longest prefix suffix) dalam algoritma KMP.

3. **Tuple**, merupakan struktur data yang menyimpan berbagai jenis elemen dalam satu variabel. Pada program ini, tuple digunakan untuk menyimpan informasi tentang karakter yang dibandingkan selama proses pencarian pola, seperti indeks dalam teks, indeks dalam pola, karakter dari teks, dan karakter dari pola.
4. **Bitmap**, merupakan struktur data yang digunakan untuk menyimpan gambar. Pada program ini, Bitmap digunakan untuk menyimpan dan memanipulasi gambar sidik jari dalam proses konversi gambar menjadi representasi biner dan ASCII.
5. **StringBuilder**, merupakan struktur data yang efisien untuk memanipulasi string secara dinamis. Pada program ini, StringBuilder digunakan untuk membangun representasi biner dari gambar sidik jari dengan cara yang efisien.
6. **Dictionary**, merupakan struktur data yang menyimpan pasangan kunci-nilai. Pada program ini, Dictionary digunakan dalam konversi teks Alay menjadi teks asli dengan menggantikan karakter-karakter tertentu dengan karakter yang sesuai.
7. **Stopwatch**, merupakan struktur data yang digunakan untuk mengukur waktu eksekusi suatu proses. Pada program ini, Stopwatch digunakan untuk mengukur waktu yang diperlukan dalam proses pencocokan sidik jari.
8. **NpgsqlConnection**, **NpgsqlCommand**, dan **NpgsqlDataReader**, digunakan untuk menghubungkan, mengeksekusi perintah, dan membaca data dari database PostgreSQL.

4.1.2 Fungsi dan Prosedur

Dalam pengembangan program ini, dibuat kelas-kelas yang mengimplementasikan struktur data tersebut. Berikut merupakan penjelasan dari setiap kelas yang dibuat.

Tabel 4.1.2.1 Kelas BMAlgorithm

Nama Kelas : BMAlgorithm
Atribut

badChar	Array yang menyimpan indeks terakhir kemunculan setiap karakter dalam pola.
Method	
BoyerMooreSearch (string pat, string txt)	Mencari kemunculan pola dalam teks menggunakan algoritma Boyer-Moore.
BadCharHeuristic (string str, int size, int[] badChar)	Mengisi array badChar dengan indeks terakhir kemunculan setiap karakter dalam pola.

Tabel 4.1.2.2 Kelas FingerprintConverter

Nama Kelas : FingerprintConverter	
Method	
ConvertImageToBinary (System.Drawing.Bitmap image)	Mengonversi gambar System.Drawing.Bitmap menjadi string biner.
ConvertImageToBinary (Avalonia.Media.Imaging.Bitmap image)	Mengonversi gambar Avalonia.Media.Imaging.Bitmap menjadi string biner.
ConvertImageToBinaryInternal (System.Drawing.Bitmap image)	Implementasi internal untuk mengonversi System.Drawing.Bitmap menjadi string biner.
ConvertBinaryToAscii (string binaryString)	Mengonversi string biner menjadi string ASCII.
CleanPattern (string source, string pattern)	Membersihkan pola dalam string sumber dengan menghapus kemunculan pertama dan terakhir dari pola.
RemoveFirstLongestOccurrence (string source, string pattern)	Menghapus kemunculan pertama terpanjang dari pola dalam string sumber.
RemoveLastLongestOccurrence (string source, string pattern)	Menghapus kemunculan terakhir terpanjang dari pola dalam string sumber.
ConvertAvaloniaBitmapToDrawingBi	Mengonversi gambar

tmap (Avalonia.Media.Imaging.Bitmap avaloniaBitmap)	Avalonia.Media.Imaging.Bitmap menjadi System.Drawing.Bitmap.
ConvertDrawingBitmapToAvaloniaBi tmap (System.Drawing.Bitmap drawingBitmap)	Mengonversi gambar System.Drawing.Bitmap menjadi Avalonia.Media.Imaging.Bitmap.

Tabel 4.1.2.3 Kelas KMPAlgorithm

Nama Kelas : KMPAlgorithm	
Method	
KMPSearch (string pat, string txt)	Mencari kemunculan pola dalam teks menggunakan algoritma KMP.
ComputeLPSArray (string pat, int M, int[] lps)	Menghitung array LPS (Longest Prefix Suffix) yang digunakan dalam algoritma KMP untuk mencocokkan pola dengan teks

Tabel 4.1.2.4 Kelas Processing

Nama Kelas : Processing	
Atribut	
Biodata Class: NIK Nama TempatLahir TanggalLahir JenisKelamin GolonganDarah Alamat Agama StatusPerkawinan Pekerjaan Kewarganegaraan	Merepresentasikan informasi biodata seseorang dengan atribut seperti NIK, Nama, TempatLahir, TanggalLahir, JenisKelamin, GolonganDarah, Alamat, Agama, StatusPerkawinan, Pekerjaan, dan Kewarganegaraan.
Method	
ProcessFingerprintMatching (Bitmap inputImage, bool algorithmChoice)	Memproses pencocokan sidik jari antara gambar input dan database menggunakan algoritma KMP atau Boyer-Moore.
LoadImageFilesFromDatabaseAsync())	Memuat daftar berkas citra dan nama pemiliknya dari database secara asinkron.

LoadBiodataAsync()	Memuat daftar biodata dari database secara asinkron.
--------------------	--

Tabel 4.1.2.5 Kelas RegexString

Nama Kelas : RegexString	
Method	
ConvertAlayToOriginal(string input)	Mengonversi teks alay menjadi bentuk aslinya.
ReplaceAll (string input, List<(string pattern, string replacement)> replacements)	Mengganti semua pola dalam string dengan penggantinya.
Replace (string input, string pattern, string replacement)	Mengganti pola pertama yang ditemukan dalam string dengan penggantinya.
RemoveNonAlphanumeric (string input)	Menghapus karakter non-alfanumerik dari string.
CapitalizeProperly (string input)	Mengubah huruf pertama setiap kata dalam string menjadi huruf besar.
FindBestMatch (string input, List<string> names, double similarityThreshold, out double highestSimilarity)	Mencari kecocokan terbaik antara input dan daftar nama berdasarkan ambang kesamaan.
CalculateSimilarity (string source, string target)	Menghitung kesamaan antara dua string menggunakan jarak Levenshtein.
ComputeLevenshteinDistance (string source, string target)	Menghitung jarak Levenshtein antara dua string.

Sementara itu, dalam pembuatan Graphical User Interface (GUI), terdiri dari file-file berikut. Views menangani tampilan dan styling pada GUI, sementara ViewModels menjadi referensi binding variable yang terdapat pada tampilan program.

Tabel 4.1.2.6 File pendukung ViewModels dan Views

ViewModels	Views	Deskripsi
MainWindowViewModel.cs	MainWindow.axaml MainWindow.axaml.cs	Menangani tampilan awal homepage program
BioPrintViewModel.cs	BioPrint.axaml BioPrint.axaml.cs	Menangani tampilan utama program dalam pencarian berkas citra sidik jari

4.2 Tata Cara Penggunaan Program

Berikut merupakan bentuk program ketika baru dibuka. Fitur yang tersedia adalah sebagai berikut:



Gambar 4.2.1. Tampilan Awal Program

Fitur-fitur yang terdapat pada program yaitu sebagai berikut:

1. *Button* “Pilih Citra” yang mengarahkan pengguna untuk memilih file sidik-jari yang ingin dicocokkan
2. *Textbox* berisi gambar file yang dipilih. Jika file tidak ditemukan atau format file peta salah, maka akan menampilkan pesan kesalahan.
3. *Toggle Switch* pemilihan algoritma antara KMP ataupun BM

4. Button “Back” yang mengarahkan pengguna ke *homepage* atau tampilan awal



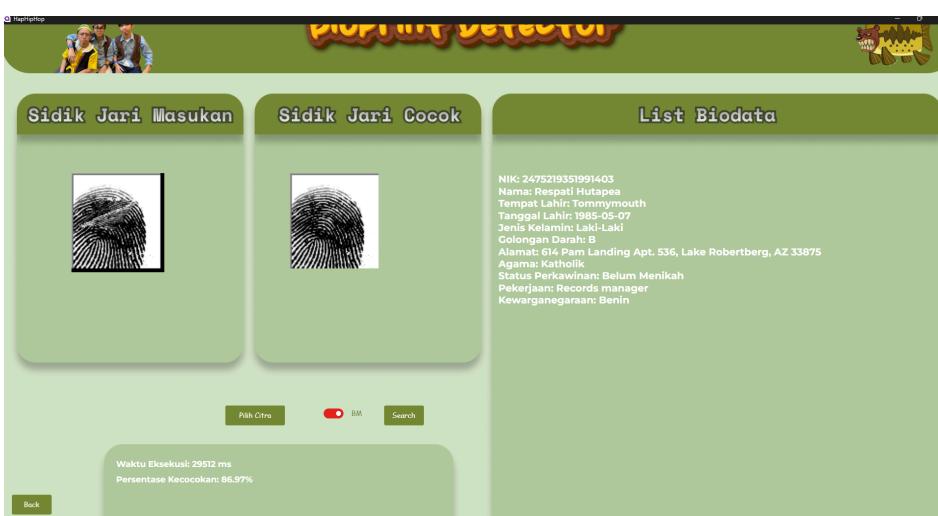
Gambar 4.2.2. Tampilan Penyelesaian Program

Berikut merupakan tampilan akhir hasil penyelesaian, memuat gambar sidik jari yang cocok, biodata dari pemilik sidik jari, waktu eksekusi serta persentase kemiripan.

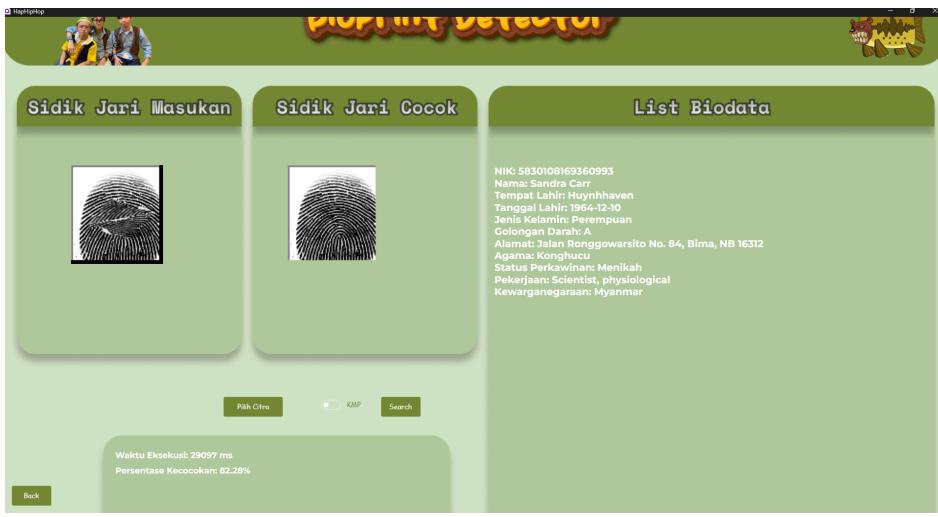
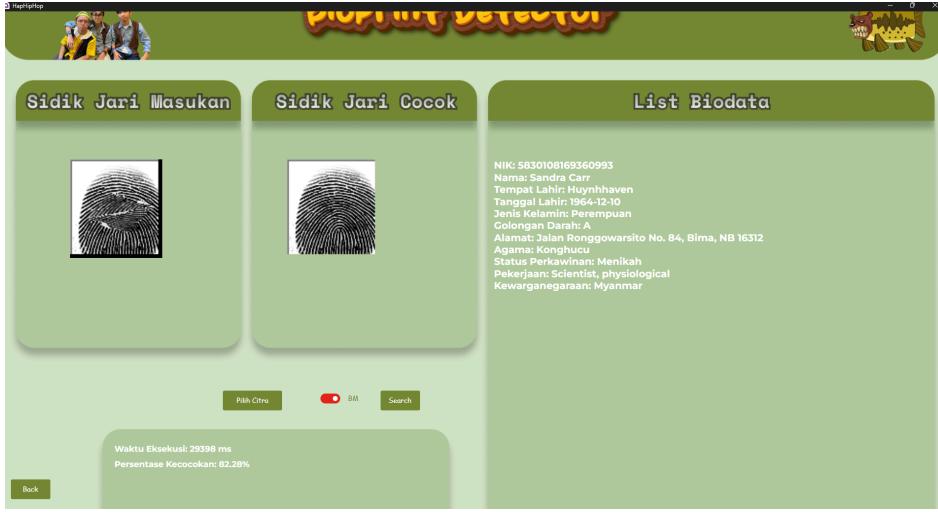
4.3 Hasil Pengujian

4.3.1 Test Case 1

Tabel 4.3.1 Hasil dari test case 1

Test Case 1	
Algoritma	Hasil
KMP	 <p>The screenshot shows the BioPrint software interface. At the top, there's a header with the BioPrint logo and a search bar. Below it, there are three main sections: 'Sidik Jari Masukan' (Input Fingerprint) showing a live video feed of a person's hand, 'Sidik Jari Cocok' (Matched Fingerprint) showing a stored fingerprint image, and 'List Biodata' (Biodata List) which displays the following information:</p> <p>NIK: 2475219351991403 Nama: Respati Hutapea Tempat Lahir: Tommymouth Tanggal Lahir: 1985-05-07 Jenis Kelamin: Laki-Laki Golongan Darah: B Alamat: G14 Pam Landing Apt. 536, Lake Robertberg, AZ 33875 Agama: Katholik Status Perkawinan: Belum Menikah Pekerjaan: Records manager Kewarganegaraan: Benin</p> <p>At the bottom of the interface, there are buttons for 'Pilih Citra' (Select Image), 'KMP' (algorithm selection), and 'Search'. A status bar at the bottom indicates 'Waktu Eksekusi: 29952 ms' and 'Persentase Kecocokan: 86.97%'.</p>
BM	 <p>The screenshot shows the BioPrint software interface. At the top, there's a header with the BioPrint logo and a search bar. Below it, there are three main sections: 'Sidik Jari Masukan' (Input Fingerprint) showing a live video feed of a person's hand, 'Sidik Jari Cocok' (Matched Fingerprint) showing a stored fingerprint image, and 'List Biodata' (Biodata List) which displays the following information:</p> <p>NIK: 2475219351991403 Nama: Respati Hutapea Tempat Lahir: Tommymouth Tanggal Lahir: 1985-05-07 Jenis Kelamin: Laki-Laki Golongan Darah: B Alamat: G14 Pam Landing Apt. 536, Lake Robertberg, AZ 33875 Agama: Katholik Status Perkawinan: Belum Menikah Pekerjaan: Records manager Kewarganegaraan: Benin</p> <p>At the bottom of the interface, there are buttons for 'Pilih Citra' (Select Image), 'BM' (algorithm selection), and 'Search'. A status bar at the bottom indicates 'Waktu Eksekusi: 29512 ms' and 'Persentase Kecocokan: 86.97%'.</p>

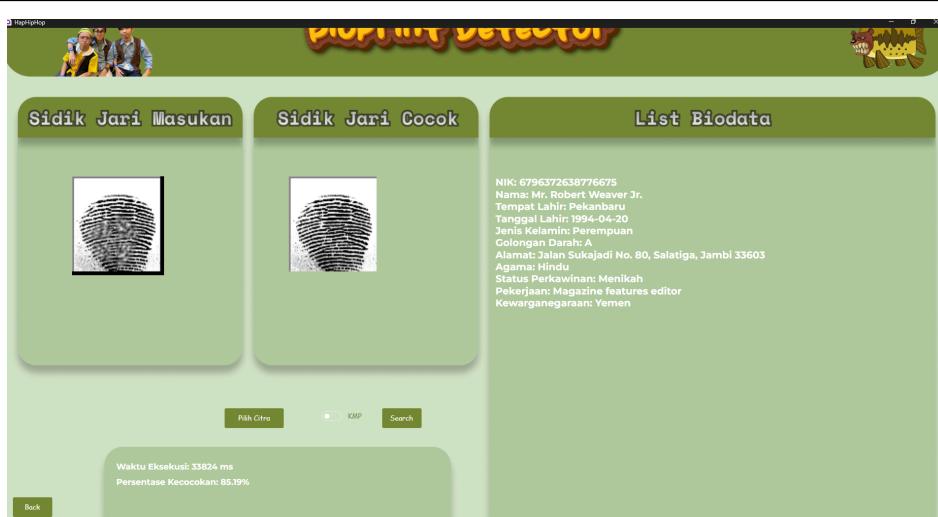
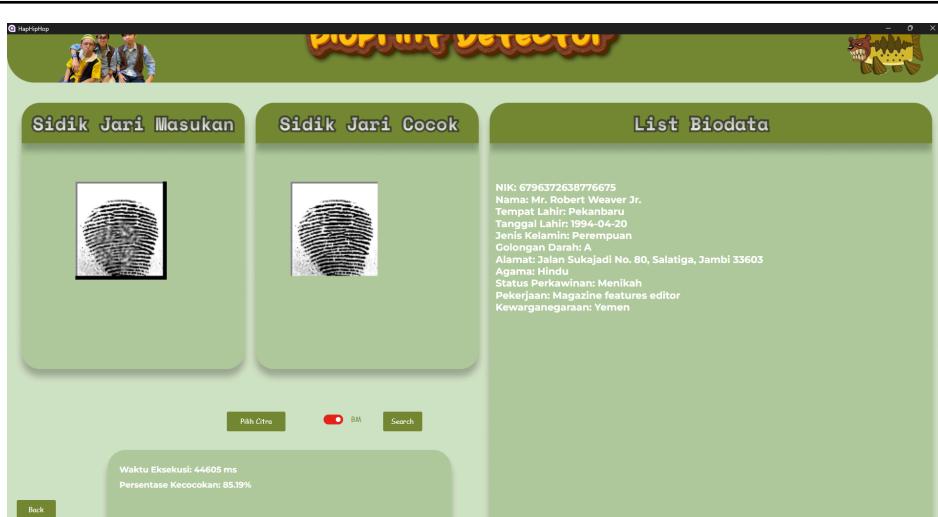
4.3.2 Test Case 2*Tabel 4.3.2 Hasil dari test case 2*

Test Case 2	
Algoritma	Hasil
KMP	 <p>The screenshot shows the BioPrint Detector software interface. The main area displays two fingerprint images: 'Sidik Jari Masukan' (Input Fingerprint) and 'Sidik Jari Cocok' (Matched Fingerprint). Below these are execution time and matching percentage details: 'Waktu Eksekusi: 29097 ms' and 'Persentase Kecocokan: 82.28%'. At the bottom left is a 'Back' button, and at the bottom right are 'Pilih Citra', 'KMP', and 'Search' buttons. The top right corner shows a user profile icon.</p>
BM	 <p>The screenshot shows the BioPrint Detector software interface. The main area displays two fingerprint images: 'Sidik Jari Masukan' (Input Fingerprint) and 'Sidik Jari Cocok' (Matched Fingerprint). Below these are execution time and matching percentage details: 'Waktu Eksekusi: 29398 ms' and 'Persentase Kecocokan: 82.28%'. At the bottom left is a 'Back' button, and at the bottom right are 'Pilih Citra', 'BM', and 'Search' buttons. The top right corner shows a user profile icon.</p>

4.3.3 Test Case 3*Tabel 4.3.3 Hasil dari test case 3*

Test Case 3	
Algoritma	Hasil
KMP	 <p>The screenshot shows the BioPrint Fingerprint Detector interface. On the left, there are two green rounded rectangular boxes: 'Sidik Jari Masukan' containing a black and white fingerprint image and 'Sidik Jari Cocok' containing a blurred version of the same fingerprint. In the center, there is a large green box labeled 'List Biodata' containing a sample record. The record includes NIK: 3281992917461416, Name: Michelle Montgomery, Tempat Lahir: East Antonio, Tanggal Lahir: 1988-06-29, Jenis Kelamin: Perempuan, Golongan Darah: AB, Alamat: Jl. Tubagus Ismail No. 3, Pagaralam, Jawa Tengah 72673, Agama: Konghucu, Status Perkawinan: Belum Menikah, and Pekerjaan: Marine scientist. Below the list is a note: 'Kewarganegaraan Republik Rakyat Tiongkok'. At the bottom of the central area, it says 'Waktu Eksekusi: 30916 ms' and 'Persentase Kecocokan: 80.66%'. A 'Back' button is at the bottom left.</p>
BM	 <p>The screenshot shows the BioPrint Fingerprint Detector interface for the BM algorithm. It has the same layout as the KMP screenshot, with 'Sidik Jari Masukan' and 'Sidik Jari Cocok' boxes, a 'List Biodata' section with the same sample record, and performance metrics at the bottom. The 'Waktu Eksekusi' is 31452 ms and the 'Persentase Kecocokan' is 80.66%. A 'Back' button is at the bottom left.</p>

4.3.4 Test Case 4*Tabel 4.3.4 Hasil dari test case 4*

Test Case 4	
Algoritma	Hasil
KMP	 <p>Waktu Eksekusi: 33824 ms Persentase Kecocokan: 85.19%</p>
BM	 <p>Waktu Eksekusi: 44605 ms Persentase Kecocokan: 85.19%</p>

4.4 Analisis Hasil Pengujian

Algoritma KMP adalah algoritma pencocokan string yang efisien untuk mencari kecocokan pola dalam teks dengan waktu kompleksitas $O(n + m)$, di mana n adalah panjang teks dan m adalah panjang pola yang dicari. Algoritma KMP tidak pernah mundur dalam teks input, sehingga lebih efisien dalam menangani data dalam jumlah besar karena tidak perlu melakukan operasi mundur. Akan tetapi, pada kasus pencocokan sidik jari, algoritma KMP tidak berfungsi dengan baik saat citra sidik jari dengan teks ASCII 8-Bits memiliki ukuran meningkat. Semakin besar ukuran alfabet, semakin besar peluang terjadinya ketidakcocokan (mismatch), karena ada lebih banyak kemungkinan ketidakcocokan. Ketidakcocokan biasanya terjadi di awal pola.

Algoritma Boyer-Moore memiliki waktu eksekusi terburuk sebesar $O(nm + A)$, di mana n adalah panjang teks dan m adalah panjang pola, sedangkan A adalah ukuran alfabet. Algoritma ini bekerja sangat cepat ketika ukuran alfabet (A) besar, namun kinerjanya menurun ketika ukuran alfabet kecil. Pada kasus pencocokan sidik jari, algoritma Boyer-Moore sangat baik digunakan untuk citra sidik jari dengan teks ASCII 8-Bits yang memiliki alfabet besar, tetapi kurang efektif untuk pencarian citra sidik jari dengan teks ASCII 8-Bits yang memiliki alfabet kecil.

Pada kasus pencocokan sidik jari, algoritma Boyer-Moore sedikit lebih cepat karena mampu menangani pencarian sidik jari dengan teks ASCII 8-Bits yang memiliki alfabet besar. Di sisi lain, algoritma KMP lebih cepat dalam menangani ketidakcocokan yang terjadi lebih belakangan dalam pola. Namun, meskipun KMP efisien dalam menangani ketidakcocokan yang terjadi di bagian akhir pola, kinerjanya menurun saat ukuran alfabet bertambah besar.

4.5 Penjelasan Implementasi Bonus Enkripsi Data

Pada implementasi enkripsi data yang kami lakukan, kami menggunakan algoritma AES. Advanced Encryption Standard (AES) adalah algoritma kriptografi simetris yang diadopsi sebagai standar enkripsi oleh pemerintah Amerika Serikat dan digunakan secara luas di seluruh dunia untuk mengamankan data digital. Berikut adalah langkah-langkah enkripsi yang dilakukan dalam program yang kami buat.

nik [PK] character varying (200)	nama character varying (255)	tempat_lahir character varying (255)
QTGOLpk+1gIIICcGNdCkig==	HL07N5U9xwfMfYkG0+p1lw9LUz+YVeltkGgUPFHxxxy...	uRkkI5eUEMrCaBQ8UfHHIQ==
0LNjPyIGtExhy6FVTQTRag==	HHrDwVpWx9YcUwgGrdlW8xfmCcoVeItkGgUPFHxxxy...	QpLhD6CjFWmQaBQ8UfHHIQ==
QY8wKClmZylxMgE0ptA0XA...	nI9PqH5mxxOs2Wta0/HHIQ==	VxmprgvwsrdZmBQ8UfHHIQ==
qt0yRJmlZwk4agHSojU08...	QYCva9Smx9+phxCjVvHHIQ==	dKeZf1NWHMqXQRQ8UfHHIQ==
C7PSmNS3m6OioQQQJyJW...	MmLhmAvkR1yphob88UfHHIQ==	Rxm0z5ec4i2QaBQ8UfHHIQ==
nTkwmJk9qCQlhzMmpiqFK...	XmbSsguRHFP6Mwjw/KSa2TxQ5hePH0yHtScPFHxxxy...	Vxmprgu+EAesHKueUfHHIQ==
qo+ytlUWaiiQxUzMo1aElKg=	0Et5ZLScigdSx9u1pg0FoQ==	96gUrZs/Gq2saBQ8UfHHIQ==
C2Ld1bS3PKM6pdVVJzXRig...	n/FDib2aYzo6xwgGsyLHIQ==	UDsMA+dnZi2QaBQ8UfHHIQ==
jFs4RGU1PKe4aqSQkd/eeQ==	zLPStX48q0Ex9iEg7kHRKDJBmCcoVeItkGgUPFHxxxyE=	KKfhGFWmAK31GBA8UfHHIQ==
OlvSN9SaZyQDAKQGBzXRa...	QdNjClFmcjq4gX4gDtCSOul4gLWZSultkGgUPFHxxxyE=	Qhkfd04a4i2QaBQ8UfHHIQ==

Gambar 4.5.1 Tampilan data pada tabel biodata saat setelah di enkripsi

A. Komponen-Komponen Utama

1. SubBytes dan InvSubBytes

Fungsi `SubBytes` dan `InvSubBytes` menggunakan Substitution Box (S-box) dan Inverse Substitution Box (invSBox) untuk melakukan substitusi byte.

- `SubBytes`: Mengganti setiap byte pada state dengan nilai yang sesuai dari S-box.
- `InvSubBytes`: Mengganti setiap byte pada state dengan nilai yang sesuai dari invSBox.

2. AddRoundKey

Fungsi `AddRoundKey` melakukan operasi XOR antara state (data yang dienkripsi) dengan kunci enkripsi pada setiap byte.

3. PadRight

Fungsi `PadRight` digunakan untuk mem-padding data yang kurang dari panjang blok yang diharapkan (16 byte) dengan nol di bagian akhir.

B. Proses Enkripsi

1. Encrypt

Fungsi `Encrypt` melakukan enkripsi pada plaintext menggunakan kunci yang diberikan. Berikut adalah langkah-langkahnya:

- Key Preparation: Kunci diubah menjadi 16 byte dengan padding.
- Block Processing: Data plaintext dipecah menjadi blok-blok 16 byte.
- Initial AddRoundKey: Blok plaintext di-XOR dengan kunci.
- SubBytes: Setiap byte dalam blok diganti dengan nilai dari S-box.
- Final AddRoundKey: Blok yang telah di-substitusi di-XOR lagi dengan kunci.
- Base64 Encoding: Data hasil enkripsi diubah menjadi string base64 untuk memudahkan penyimpanan dan transfer data.

2. Decrypt

Fungsi `Decrypt` melakukan dekripsi pada ciphertext menggunakan kunci yang diberikan. Berikut adalah langkah-langkahnya:

- Base64 Decoding: Ciphertext diubah kembali dari string base64 menjadi byte array.
- Block Processing: Data ciphertext dipecah menjadi blok-blok 16 byte.
- Initial AddRoundKey: Blok ciphertext di-XOR dengan kunci.
- InvSubBytes: Setiap byte dalam blok diganti dengan nilai dari invSBox.
- Final AddRoundKey: Blok yang telah di-substitusi di-XOR lagi dengan kunci.
- Padding Removal: Data hasil dekripsi dihilangkan padding-nya.

C. Enkripsi khusus untuk tipe tanggal

1. EncryptDate dan DecryptDate

Fungsi `EncryptDate` dan `DecryptDate` digunakan untuk mengenkripsi dan mendekripsi tanggal dalam format YYYY-MM-DD.

- EncryptDate: Mengubah objek tanggal menjadi string, kemudian mengenkripsi string tersebut menggunakan Encrypt.
- DecryptDate: Mendekripsi string terenkripsi menjadi string tanggal, kemudian mengubahnya kembali menjadi objek tanggal.

D. Penyimpanan Data

1. Penyimpanan Data

Dalam fungsi `seed_data`, data dienkripsi sebelum disimpan ke dalam basis data PostgreSQL. Setiap data sensitif seperti NIK, nama, tempat lahir, tanggal lahir, golongan darah, alamat, agama, pekerjaan, dan kewarganegaraan dienkripsi menggunakan fungsi `Encrypt` dan `EncryptDate`.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar III IF2211 Strategi Algoritma ini telah diimplementasikan algoritma KMP dan BM beserta kelas-kelas pendukung dalam tujuan untuk menyelesaikan permasalahan *pattern matching* yang tertera pada spek. Kelas-kelas pendukung mencakup hal yang menangani GUI (Graphical User Interface) menggunakan framework Avalonia, kelas struktur data, dan kelas interface. Pada hasil pengujian sebelumnya terlihat bahwa algoritma KMP dan BM selalu berhasil menyelesaikan permasalahan yang valid. Algoritma BM cenderung lebih cepat daripada algoritma KMP. Dengan demikian, penulis menyimpulkan bahwa melalui Tugas Besar III IF2211 Strategi Algoritma ini, dapat dibuat sebuah algoritma berbasis KMP dan BM diimplementasikan dalam suatu program yang mengkomputasi penyelesaian *pattern matching*.

5.2 Saran

Tugas Besar III IF2211 Strategi Algoritma Semester II Tahun 2023/2024 menjadi salah satu tugas yang memberikan pelajaran baru bagi penulis. Berdasarkan pengalaman penulis mengerjakan tugas ini, berikut merupakan saran untuk pembaca yang ingin melakukan atau mengerjakan hal yang serupa.

1. Keefektifan dalam kerja sama tim merupakan hal yang penting dalam mengerjakan tugas ini. Selain pembagian tugas yang merata, penulis terbantu oleh beberapa kali kerja kelompok.
2. Perancangan algoritma dan struktur program perlu diperhatikan dalam mengerjakan tugas ini. Implementasi struktur program yang tepat akan memudahkan dalam pembuatan Graphical User Interface (GUI) dikarenakan memerlukan banyak binding

5.3 Refleksi

Tugas Besar III IF2211 Strategi Algoritma merupakan salah satu tugas besar yang memerlukan usaha lebih dari biasanya. Meskipun demikian, kami mengapresiasi tugas

besar ini karena memberikan kami kesempatan untuk bereksplorasi dan mengimplementasikan algoritma yang menarik.

5.4 Tanggapan

Pada Tugas Besar III IF2211 ini, kami sadar bahwa hasil yang telah kami kerjakan masih memiliki celah untuk improvisasi dalam beberapa hal. Secara keseluruhan, Tugas Besar ini tergolong menyenangkan bagi penulis karena dapat mengeksplorasi pengembangan algoritma pencarian KMP dan BM, mengeksplorasi kemungkinan-kemungkinan edge case dari pencarian data menggunakan *pattern matching* dan juga men-develop sebuah aplikasi berbasis desktop menggunakan Bahasa C#. Tanggapan terakhir terkait Tugas Besar ini adalah Tugas Besar ini memiliki persoalan yang menarik, dan juga menantang penulis untuk lebih mendalami algoritma *string matching*.

DAFTAR PUSTAKA

- Munir, R. (2021). Strategi Algoritma: Pencocokan String (String/Pattern Matching).
Dilansir dari Homepage Rinaldi Munir:
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>. Diakses pada 9 Juni 2024.
- Munir, R. (2019). Strategi Algoritma: String Matching dengan Regular Expression.
Dilansir dari Homepage Rinaldi Munir:
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>. Diakses pada 9 Juni 2024.

LAMPIRAN

Repository

Link Repository dari Tugas Besar 3 IF2211 Strategi Algoritma kelompok 7 “HapHipHop” adalah sebagai berikut.

https://github.com/zultopia/Tubes3_HapHipHop.git

Youtube

Link video Youtube dari Tugas Besar 3 IF2211 Strategi Algoritma kelompok 7 “HapHipHop” adalah sebagai berikut.

<https://youtu.be/Huryu5GguwE?si=mNOTxMB90MjoDcfI>