

**Laporan Tugas Besar 3  
IF2211 Strategi Algoritma**

**Pemanfaatan *Pattern Matching* dalam Membangun Sistem  
Deteksi Individu Berbasis  
Biometrik Melalui Citra Sidik Jari**

**Semester II Tahun 2023/2024**



Disusun Oleh:  
Dewantoro Triatmojo 13522011  
Nyoman Ganadipa Narayana 13522066  
Randy Verdian 13522067

**Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2024**

# Daftar Isi

<b>Daftar Isi</b>	<b>2</b>
<b>Bab 1</b>	
<b>Deskripsi Tugas</b>	<b>4</b>
<b>Bab 2</b>	
<b>Landasan Teori</b>	<b>6</b>
2.1. Algoritma KMP, BM, Regex	6
2.1.1 KMP (Knuth Morris Pratt)	6
2.1.2. BM (Boyer Moore)	6
2.1.3. Regex	7
2.2. Pengukuran Kemiripan	7
2.3. GoyangSinggalang Fingerprint Apps	8
<b>Bab 3</b>	
<b>Analisis Pemecahan Masalah</b>	<b>9</b>
3.1. Penyelesaian Masalah	9
3.1.1. Konversi Gambar ASCII 8 Bit	9
3.1.2. Algoritma Program	9
3.2. Penyelesaian Solusi dengan Algoritma KMP, BM, dan LCS	10
3.2.1 Algoritma KMP	10
3.2.2 Algoritma BM	11
3.2.3 Algoritma Levenshtein Distance	12
3.2.4 Algoritma AES	12
3.2.5 Implementasi Regex Generator	13
3.3. Fitur Fungsional dan Arsitektur Aplikasi	13
3.3.1 Fitur Fungsional	13
3.3.2 Arsitektur Aplikasi	14
3.4. Contoh Ilustrasi Kasus	14
<b>Bab 4</b>	
<b>Implementasi dan Pengujian</b>	<b>17</b>
4.1. Spesifikasi Teknis Program	17
4.1.1 Implementasi KMP	17
4.1.2 Implementasi BM	18
4.1.3 Struktur Data Fingerprint	18
4.1.4 Struktur Data User	19
4.1.4 Struktur Data Db	21
4.1.5 Namespace lib	25
4.1.5.1 Struktur data ImageConverter	25
4.1.5.2 Implementasi Algoritma AES	26
4.1.6 Program Utama	28
4.1.6.1 Hubungan GUI dengan Algoritma	28
4.1.6.2 Method solve pada Solver yang diimplementasikan	30

4.2. Penggunaan Program	33
4.2.1 Setup Database MySQL	33
4.2.2 Menjalankan Program	34
4.2.3 Menggunakan Program	34
4.3. Uji Coba Program	36
4.3.1 Uji Coba Program	36
4.3.1.1 Fingerprint Real	36
4.3.1.2 Fingerprint Altered Easy	38
4.3.1.3 Fingerprint Altered Medium	40
4.3.1.4 Fingerprint Altered Hard	42
4.3.2 Uji Coba Stress	43
4.4. Analisis Hasil Pengujian	44
<b>Bab 5</b>	
<b>Kesimpulan dan Saran</b>	<b>45</b>
5.1 Kesimpulan	45
5.2 Saran	45
<b>Lampiran</b>	<b>46</b>
<b>Daftar Pustaka</b>	<b>47</b>

# Bab 1

## 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, kami 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 2

# Landasan Teori

### 2.1. Algoritma KMP, BM, Regex

#### 2.1.1 KMP (Knuth Morris Pratt)

Algoritma Knuth-Morris-Pratt (KMP) adalah metode pencocokan string yang efisien untuk menemukan sebuah substring dalam sebuah string teks. Algoritma ini pertama-tama membangun tabel Longest Prefix Suffix (LPS) untuk pola yang diberikan, yang menyimpan informasi tentang panjang prefiks yang juga merupakan sufiks terpanjang untuk setiap posisi dalam pola. Tabel LPS ini membantu menghindari pemeriksaan ulang karakter yang sudah dibandingkan sebelumnya. Setelah tabel LPS dibangun, algoritma kemudian mencocokkan pola dengan teks dengan membandingkan karakter satu per satu. Jika terjadi ketidakcocokan, tabel LPS digunakan untuk menentukan posisi berikutnya dari pola yang perlu diperiksa, sehingga tidak perlu mencocokkan ulang karakter yang sudah diperiksa. Algoritma KMP berjalan dalam waktu linier  $O(n + m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola, membuatnya sangat efisien karena setiap karakter dalam teks hanya diperiksa sekali. Dengan menggunakan tabel LPS untuk menghindari perulangan, algoritma ini lebih efisien dibandingkan metode brute force, dan sangat berguna dalam aplikasi yang memerlukan pencarian substring yang cepat, seperti editor teks dan alat pencarian.

#### 2.1.2. BM (Boyer Moore)

Algoritma Boyer-Moore (BM) adalah metode pencocokan string yang sangat efisien untuk menemukan sebuah substring dalam sebuah string teks, diperkenalkan oleh Robert S. Boyer dan J Strother Moore pada tahun 1977. Algoritma ini memanfaatkan dua tabel penting: tabel bad character dan tabel good suffix. Tabel bad character mencatat posisi terakhir dari setiap karakter dalam pola, memungkinkan penggeseran pola melewati posisi karakter yang tidak cocok jika terjadi ketidakcocokan. Tabel good suffix mencatat informasi tentang sufiks yang sudah cocok, memungkinkan pola untuk digeser agar mempertahankan sufiks yang cocok jika terjadi ketidakcocokan setelah beberapa karakter cocok. Algoritma BM mulai mencocokkan dari akhir pola dengan teks dan menggunakan kedua tabel tersebut untuk menentukan penggeseran yang optimal saat ketidakcocokan terjadi, sehingga memungkinkan melewati banyak karakter sekaligus dan mempercepat proses pencocokan. Dalam praktik, algoritma Boyer-Moore sering kali lebih cepat daripada algoritma pencocokan string lainnya, terutama untuk pola dan teks yang panjang, menjadikannya sangat berguna dalam aplikasi yang

memerlukan pencarian substring yang cepat dan efisien, seperti pemrosesan teks besar dan analisis data kompleks.

### 2.1.3. Regex

Algoritma Regex, singkatan dari Regular Expression, adalah metode yang digunakan untuk mencocokkan pola teks dengan pola yang ditentukan secara formal. Regular expression digunakan untuk mencari, mengganti, dan memanipulasi teks dengan pola tertentu. Algoritma Regex menggunakan aturan sintaks untuk menggambarkan pola yang ingin dicocokkan, seperti karakter tertentu, kelompok karakter, atau pola tertentu dari karakter. Saat diterapkan, algoritma ini mencocokkan pola tersebut dengan teks, mencari kecocokan sesuai dengan aturan yang ditetapkan. Algoritma Regex dapat sangat kuat dan fleksibel, memungkinkan pencocokan pola yang kompleks dan manipulasi teks yang canggih, seperti pencocokan pola dengan karakter khusus, kuantifikasi, grup, dan lain-lain. Namun, implementasi algoritma Regex yang efisien memerlukan pendekatan yang cermat, terutama karena kompleksitas dari ekspresi reguler yang dapat diproses. Algoritma Regex memiliki banyak aplikasi dalam pemrosesan teks, validasi input pengguna, analisis data, dan banyak lagi, menjadikannya salah satu alat yang penting dalam pengembangan perangkat lunak modern.

## 2.2. Pengukuran Kemiripan

Teknik pengukuran persentase kemiripan dalam konteks aplikasi sidik jari sering menggunakan metode yang disebut dengan *matching score* atau *similarity score*. Proses ini dimulai dengan konversi gambar sidik jari ke dalam string ASCII 8 bit. Setiap gambar sidik jari diproses dengan cara mengambil RGB dari setiap pixel gambar, kemudian dikonversi menjadi biner dengan aturan tertentu, dan akhirnya diubah menjadi ASCII 8 bit. Langkah selanjutnya adalah mencocokkan string ASCII 8 bit dari gambar sidik jari yang baru dengan data sidik jari yang ada dalam basis data.

Algoritma KMP (Knuth-Morris-Pratt) dan BM (Boyer-Moore) digunakan dalam proses pencocokan ini. Algoritma KMP digunakan untuk mencari pola yang dihasilkan dari gambar sidik jari pada teks yang dihasilkan dari basis data sidik jari. Algoritma BM, di sisi lain, mencocokkan pola dengan teks, dengan tahap preprocessing yang melibatkan inisialisasi array last occurrence dan pencarian yang memanfaatkan aturan Boyer-Moore.

Selanjutnya, jika pola ditemukan, perhitungan similarity dilakukan menggunakan algoritma Levenshtein distance untuk mendapatkan edit distance kemudian similiarity dihitung berdasarkan  $S = 1 - \frac{distance}{MAX(length)}$  untuk menentukan seberapa mirip kedua

sidik jari tersebut. Jika similarity di atas 30%, maka data sidik jari beserta identitasnya disimpan. Jika tidak ada pola yang ditemukan di atas 30% similarity, maka akan mengeluarkan error bahwa hasil tidak berhasil ditemukan.

Proses selanjutnya melibatkan pencarian identitas dari data sidik jari yang cocok pada tabel biodata. Seluruh biodata yang tersedia akan dicocokkan dengan menggunakan Regex, terutama untuk menangani kolom nama yang mungkin corrupt. Setelah nama cocok ditemukan, identitas lengkap dan sidik jari yang cocok akan dikembalikan untuk ditampilkan.

Dengan mengintegrasikan algoritma KMP, BM, dan LCS, aplikasi ini dapat mencapai tingkat akurasi yang tinggi dalam mencocokkan sidik jari dengan data pada basis data, serta menemukan identitas yang sesuai dalam tabel biodata.

### **2.3. GoyangSinggalang Fingerprint Apps**

GoyangSinggalang Fingerprint Apps adalah aplikasi desktop yang dirancang untuk mempermudah proses identifikasi individu berdasarkan sidik jari. Dengan aplikasi ini, pengguna dapat dengan mudah mengambil gambar sidik jari menggunakan perangkat input yang sesuai, seperti scanner sidik jari, yang kemudian akan diolah dan disimpan dalam basis data. Kemampuan konversi gambar sidik jari ke format yang sesuai, seperti ASCII 8 bit, memungkinkan penyimpanan data sidik jari secara efisien untuk pengolahan lebih lanjut.

Salah satu fitur utama dari GoyangSinggalang Fingerprint Apps adalah kemampuannya dalam mencocokkan gambar sidik jari yang baru diambil dengan data sidik jari yang telah tersimpan dalam basis data. Dengan menggunakan algoritma pencocokan string seperti KMP (Knuth-Morris-Pratt) dan BM (Boyer-Moore), aplikasi ini dapat mengidentifikasi individu berdasarkan sidik jari yang dimiliki. Hasil pencocokan ini akan memberikan informasi detail mengenai identitas individu tersebut.

Selain fitur pencocokan sidik jari, GoyangSinggalang Fingerprint Apps juga dilengkapi dengan kemampuan untuk memproses dan mencocokkan informasi identitas individu yang ditemukan dengan data biodata yang tersedia. Dengan teknik seperti Regex (Regular Expression), aplikasi ini dapat menangani kolom-kolom yang mungkin tidak lengkap atau tidak tepat dalam data biodata, sehingga memberikan solusi yang komprehensif dalam pengelolaan data sidik jari dan identifikasi individu dalam berbagai konteks aplikasi, mulai dari pengamanan keamanan fisik hingga manajemen kehadiran di tempat kerja.

# Bab 3

## Analisis Pemecahan Masalah

### 3.1. Penyelesaian Masalah

Tugas besar 3 Strategi Algoritma kali ini bertujuan untuk mencari identitas seseorang dari gambar sidik jari yang diberikan dengan mencocokkan masukan sidik jari dengan data sidik jari pada basis data. Gambar diolah dalam bentuk ASCII 8 bit sesuai dengan spesifikasi agar string tidak sepanjang dalam bentuk biner.

#### 3.1.1. Konversi Gambar ASCII 8 Bit

Berikut merupakan algoritma umum konversi dari gambar menjadi ascii 8 bit:

1. Ambil RGB dari suatu pixel gambar.
2. Hitung grayscale dari setiap pixel
3. Konversi menjadi biner (grayscale < 128 adalah 0 dan sisanya 1)
4. Untuk setiap 8 pixel yang sudah menjadi biner (8 bit terkumpul), ubah menjadi ASCII 8 bit.
5. Jika pada akhir gambar masih terdapat sisa pada biner yang terkumpul, tambahkan biner 0 sampai panjangnya 8 dan ubah menjadi ASCII 8 bit.

#### 3.1.2. Algoritma Program

Berikut merupakan algoritma umum program:

1. Program menerima gambar masukan sidik jari. Crop gambar agar mengambil fingerprintnya saja, dengan mengukur whitespace di setiap sisinya. Konversi gambar ini menjadi string ASCII 8 bit. String ini nantinya akan menjadi *pattern* pada string matching algoritma KMP & BM. Digunakan teknik sampling, yaitu mengambil 5 pattern pada input image yang akan digunakan sebagai pattern dalam algoritma KMP & BM.
2. Ambil seluruh gambar sidik jari pada basis data. Untuk setiap gambar yang diambil, lakukan pengecilan gambar dengan menghilangkan whitespace di setiap sisinya, kemudian konversi gambar tersebut menjadi string ASCII 8 bit. String ini menjadi text pada string matching algoritma KMP & BM. Gunakan algoritma KMP & BM untuk menentukan apakah pattern pada step 1 ditemukan pada text.
  - Jika pattern ditemukan, perhitungan diberhentikan kemudian hitung similarity menggunakan algoritma *Levenshtein Distance* dengan similarity mengikuti persamaan  $S = 1 - \frac{dist}{MAX(length)}$  dan simpan data sidik jari & nama yang cocok tersebut.

- Jika pattern tidak ditemukan, iterate ke sidik jari selanjutnya.
3. Jika tidak ada pattern yang ditemukan dari seluruh sidik jari dari basis data, maka pencarian akan menggunakan similarity tertinggi diatas 30% dengan *Levensthein* distance dengan persamaan similarity seperti pada step sebelumnya. Perhitungan similiarity antara 2 gambar digunakan teknik sampling dengan mengambil 5 pattern pada gambar 1 dan 5 pattern pada gambar 2 kemudian diambil rata-rata similiarity. Jika tidak ada similarity yang ditemukan diatas 30%, maka akan mengeluarkan error bahwa hasil tidak berhasil ditemukan.
  4. Jika ada pattern yang ditemukan, gunakan nama pada data sidik jari yang disimpan untuk mencari identitas pada tabel biodata. Ambil seluruh biodata yang tersedia. Untuk setiap biodata, cocokkan nama dengan Regex (untuk menghandle kolom nama yang *corrupt* pada tabel biodata). Setelah nama ditemukan, kembalikan identitas lengkap dan fingerprint yang cocok untuk ditampilkan.

## 3.2. Penyelesaian Solusi dengan Algoritma KMP, BM, dan LCS

Dalam tugas besar 3 Strategi Algoritma, digunakan 3 algoritma yaitu Algoritma KMP, BM, dan *Levensthein* Distance.

### 3.2.1 Algoritma KMP

Algoritma KMP digunakan untuk mencari pattern yang dihasilkan input gambar sidik jari dalam text yang dihasilkan dari basis data sidik jari. Secara umum, ada dua tahap dalam algoritma KMP yaitu preprocessing dan pencarian.

1. Pre-processing
  - a. Buat tabel prefix atau array Longest Prefix Suffix (LPS) untuk pola yang akan dicari.
  - b. Mulai dari indeks kedua pola (karena LPS untuk indeks pertama selalu 0) dan bandingkan karakter saat ini dengan karakter pertama dari pola.
  - c. Jika cocok, tingkatkan nilai LPS untuk indeks saat ini dan lanjutkan ke karakter berikutnya.
  - d. Jika tidak cocok, gunakan nilai LPS sebelumnya untuk menentukan langkah selanjutnya.
2. Pencarian

- a. Bandingkan pola dengan teks utama dari awal.
- b. Jika karakter cocok, lanjutkan ke karakter berikutnya dari teks dan pola.
- c. Jika terjadi ketidakcocokan:
  - o Jika posisi dalam pola bukan yang pertama, gunakan nilai dalam LPS Array untuk menentukan langkah selanjutnya dalam pola tanpa menggerakkan teks.
  - o Jika posisi dalam pola adalah yang pertama, geser pola ke kanan satu posisi dalam teks dan lanjutkan perbandingan.
- d. Ulangi langkah ini sampai pola ditemukan atau teks habis.

### **3.2.2 Algoritma BM**

Algoritma BM digunakan untuk mencari pattern yang dihasilkan input gambar sidik jari dalam text yang dihasilkan dari basis data sidik jari. Secara umum, ada dua tahap dalam algoritma KMP yaitu preprocessing dan pencarian.

1. Pre-processing
  - a. Inisialisasi array last occurrence
  - b. Isi array last occurrence dengan posisi terakhir kemunculan setiap karakter dalam pola
2. Pencarian
  - a. Jika panjang pola lebih besar dari panjang teks, maka pencarian dihentikan dan mengembalikan false.
  - b. Selama i lebih kecil dari panjang teks n, (i iterator pada teks dan j iterator pada pattern)
    - i. Jika karakter dalam pattern sama dengan karakter dalam teks,
      1. Jika sudah mencapai awal pola, artinya pola ditemukan dalam teks, kembalikan true.
      2. Jika belum mencapai awal pola, geser satu posisi ke kiri pada teks dan pola.
    - ii. Jika karakter dalam pattern sama dengan karakter dalam teks,
      1. Temukan posisi terakhir karakter teks yang tidak cocok dalam pola menggunakan array last\_occurrence.
      2. Geser pola ke kanan dengan jumlah yang ditentukan oleh aturan Boyer-Moore, yaitu  $m - \text{Math.Min}(j, 1 + k)$ .
      3. Kembalikan j ke posisi akhir pola.

### **3.2.3 Algoritma Levenshtein Distance**

Algoritma Levenshtein Distance digunakan untuk mencari similarity antara dua string.

1. Buat sebuah matriks 2D dp dengan ukuran  $(n+1) \times (m+1)$ .
2. Isi baris dan kolom pertama matriks dengan nilai 0 hingga n dan 0 hingga m. Ini karena jarak Levenshtein dari string apa pun dengan string kosong adalah panjang string itu sendiri.
3. Iterasi melalui setiap karakter dari kedua string (misalkan string pertama adalah X dan string kedua adalah Y).
4. Untuk setiap pasangan karakter  $(X[i], Y[j])$ :
5. Jika karakter  $X[i]$  sama dengan karakter  $Y[j]$ , maka isi matriks di posisi  $(i+1, j+1)$  dengan nilai dari posisi  $(i, j)$ . Ini menandakan bahwa tidak ada perubahan yang diperlukan.
6. Jika karakter  $X[i]$  tidak sama dengan karakter  $Y[j]$ , maka isi matriks di posisi  $(i+1, j+1)$  dengan nilai minimum antara tiga kemungkinan: penggantian karakter, penghapusan karakter, atau penambahan karakter, ditambah satu. Ini menandakan operasi minimum yang diperlukan untuk mengubah satu string menjadi string lain.
7. Setelah matriks diisi, nilai di sudut kanan bawah matriks adalah jarak Levenshtein antara kedua string.

### **3.2.4 Algoritma AES**

#### **1. Inisialisasi Kunci:**

Kunci enkripsi diambil dari variabel lingkungan PRIVATE\_KEY yang panjangnya harus 32 karakter (256 bit).

#### **2. Konversi Teks ke Byte**

Teks plaintext yang akan dienkripsi dikonversi menjadi array byte menggunakan Encoding.UTF8.GetBytes().

#### **3. Padding PKCS7:**

Jika panjang plaintext bukan kelipatan dari 16 byte, maka padding PKCS7 diterapkan untuk memastikan panjang data adalah kelipatan dari 16 byte. Fungsi ApplyPKCS7Padding menambahkan padding.

#### **4. Pembagian Blok:**

Data yang sudah dipadding kemudian dibagi menjadi blok-blok 16 byte.

#### **5. Ekspansi Kunci:**

Kunci utama diekspansi menjadi kunci round menggunakan fungsi KeyExpansion.

#### **6. Enkripsi Setiap Blok:**

Setiap blok dienkripsi secara terpisah:

- **Inisialisasi State:**
  - Data 16 byte dari blok diinisialisasi ke dalam array 4x4 byte (state).
- **AddRoundKey Awal:**
  - Kunci round awal ditambahkan ke state.
- **Ronde Enkripsi:**
  - 13 ronde enkripsi yang melibatkan operasi SubBytes, ShiftRows, MixColumns, dan AddRoundKey.
  - Pada ronde ke-14 (terakhir), hanya operasi SubBytes, ShiftRows, dan AddRoundKey dilakukan tanpa MixColumns.

#### **7. Penggabungan Hasil:**

Blok-blok terenkripsi digabungkan kembali menjadi array byte hasil enkripsi.

#### **8. Konversi ke Base64:**

Hasil enkripsi dikonversi ke dalam format Base64 untuk memudahkan penyimpanan dan pengiriman data teks.

### **3.2.5 Implementasi Regex Generator**

Dari bahasa alay, di-generate regex dengan metode sebagai berikut:

1. Untuk setiap karakter, dapat dimasukkan selipkan karakter alfabetik lain setelahnya
2. Untuk setiap karakter, karakter tersebut sebenarnya merupakan suatu huruf yang dikonversi menjadi angka karena alay atau memang angka sehingga digunakan regex OR.
3. Untuk setiap karakter karakter tersebut pada aslinya dapat berupa lowercase atau uppercase.

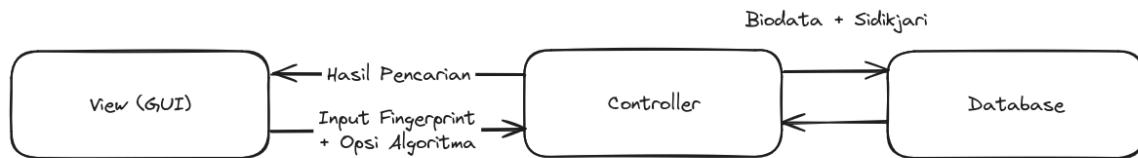
## **3.3. Fitur Fungsional dan Arsitektur Aplikasi**

### **3.3.1 Fitur Fungsional**

Program ini memiliki tiga fitur fungsional, yaitu

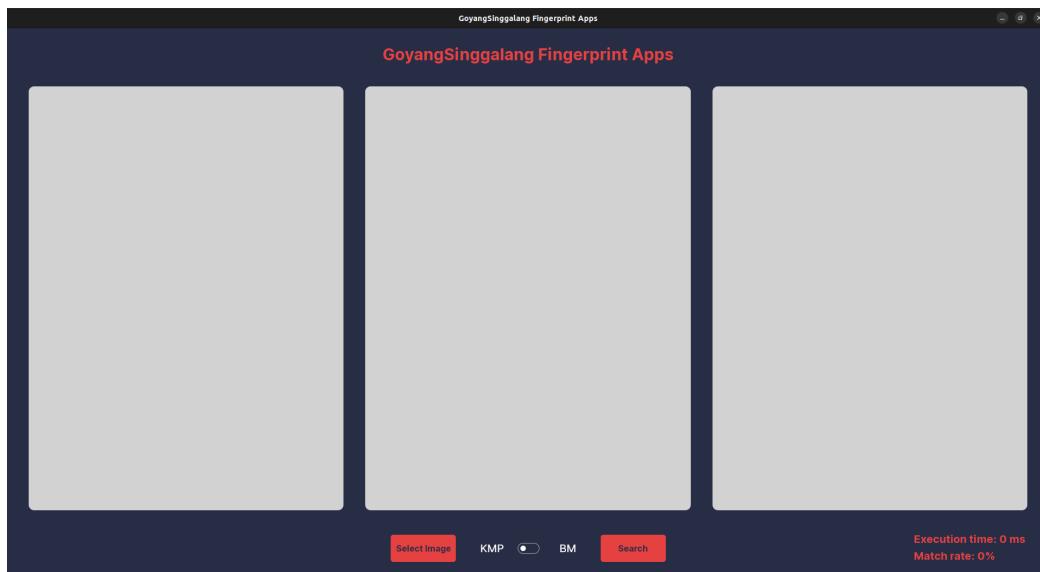
1. Pencarian sidik jari menggunakan algoritma KMP
2. Pencarian sidik jari menggunakan algoritma BM
3. Enkripsi data pengguna dengan algoritma AES

### 3.3.2 Arsitektur Aplikasi

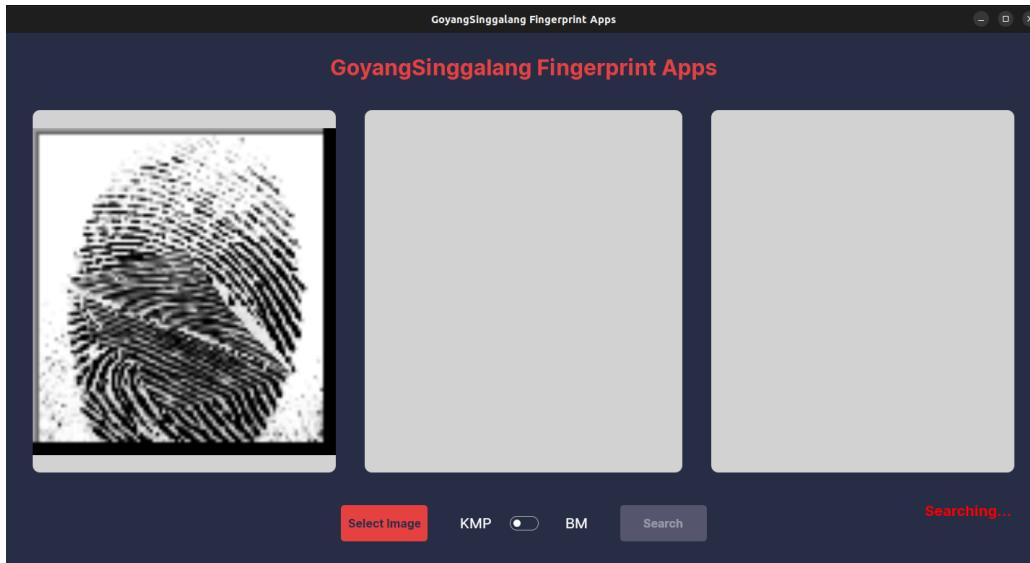


Arsitektur aplikasi pencarian relatif sederhana, yaitu terdiri dari View, Controller, dan Database. View/GUI menjadi user interface pengguna untuk memasukkan gambar sidik jari & opsi algoritma pencarian dan melihat hasil pencarian. Controller berguna untuk melakukan perhitungan pencarian sidik jari serta data biodata. Database berguna untuk menyimpan biodata orang dan juga sidik jari.

### 3.4. Contoh Ilustrasi Kasus



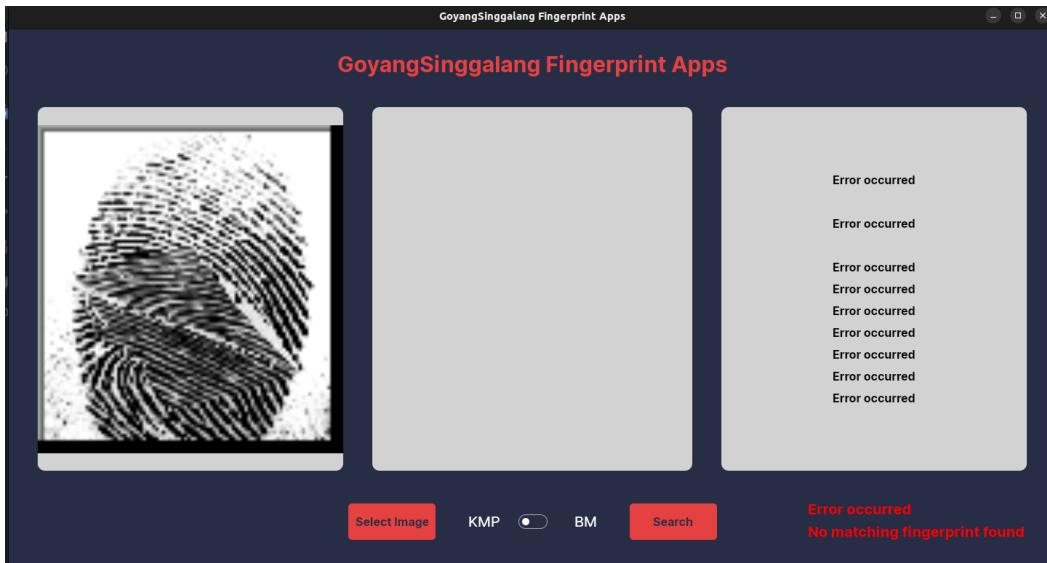
Gambar 3.4.1 Tampilan awal



Gambar 3.4.2 Tampilan loading pencarian



Gambar 3.4.3 Tampilan hasil ditemukan



**Gambar 3.4.4** Tampilan hasil tidak ditemukan

# Bab 4

## Implementasi dan Pengujian

### 4.1. Spesifikasi Teknis Program

#### 4.1.1 Implementasi KMP

KMP yang diimplementasikan mengikuti langkah-langkah yang telah dijelaskan pada 3.2.1 Tentang KMP, berikut adalah implementasi KMP pada program kami yang kami buat.

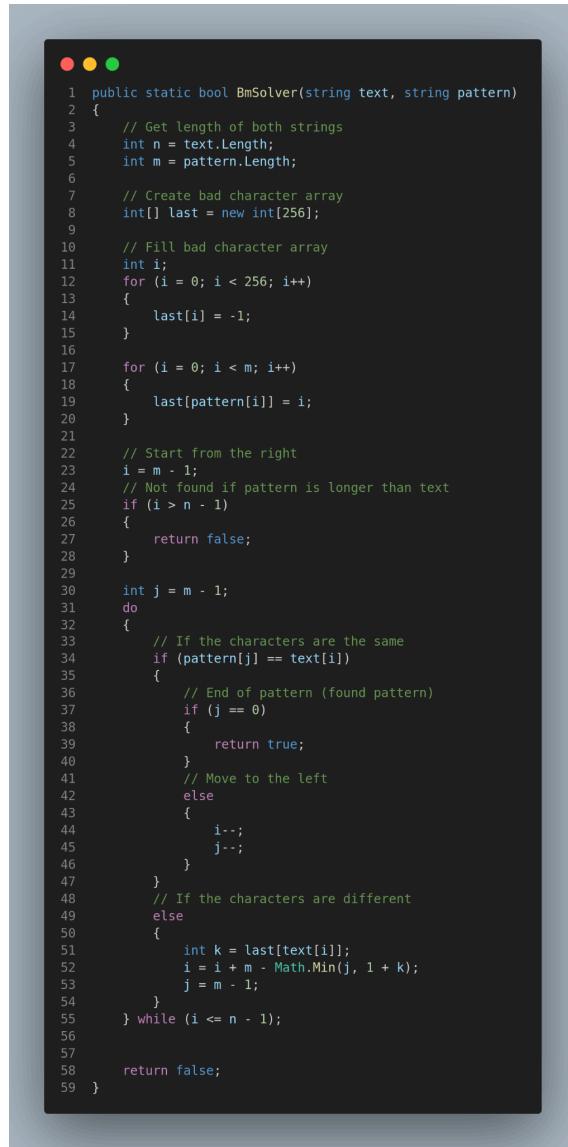


```
1  public static bool KmpSolver(string text, string pattern)
2  {
3      // Get length of both strings
4      int n = text.Length;
5      int m = pattern.Length;
6
7      // Create LPS array
8      int[] lps = new int[m];
9      lps[0] = 0;
10     int i = 1;
11     int j = 0;
12
13     while (i < n)
14     {
15         if (pattern[i] == pattern[j])
16         {
17             lps[i] = j + 1;
18             i++;
19             j++;
20         }
21         else if (j > 0)
22         {
23             j = lps[j - 1];
24         }
25         else
26         {
27             lps[i] = 0;
28             i++;
29         }
30     }
31
32     i = 0;
33     j = 0;
34     // Iterate through the text
35     while (i < n)
36     {
37         // If the characters are the same
38         if (pattern[j] == text[i])
39         {
40             // End of pattern (found pattern)
41             if (j == m - 1)
42             {
43                 return true;
44             }
45             // Move to the right
46             else
47             {
48                 i++;
49                 j++;
50             }
51         }
52         // If the characters are different
53         else if (j > 0)
54         {
55             j = lps[j - 1];
56         }
57         else
58         {
59             i++;
60         }
61     }
62
63     // Not found pattern
64     return false;
65 }
66 }
```

Gambar 4.1.1 Implementasi KMP

## 4.1.2 Implementasi BM

BM yang diimplementasikan mengikuti langkah-langkah yang telah dijelaskan pada 3.2.2 Tentang BM, berikut adalah implementasi BM pada program kami yang kami buat.

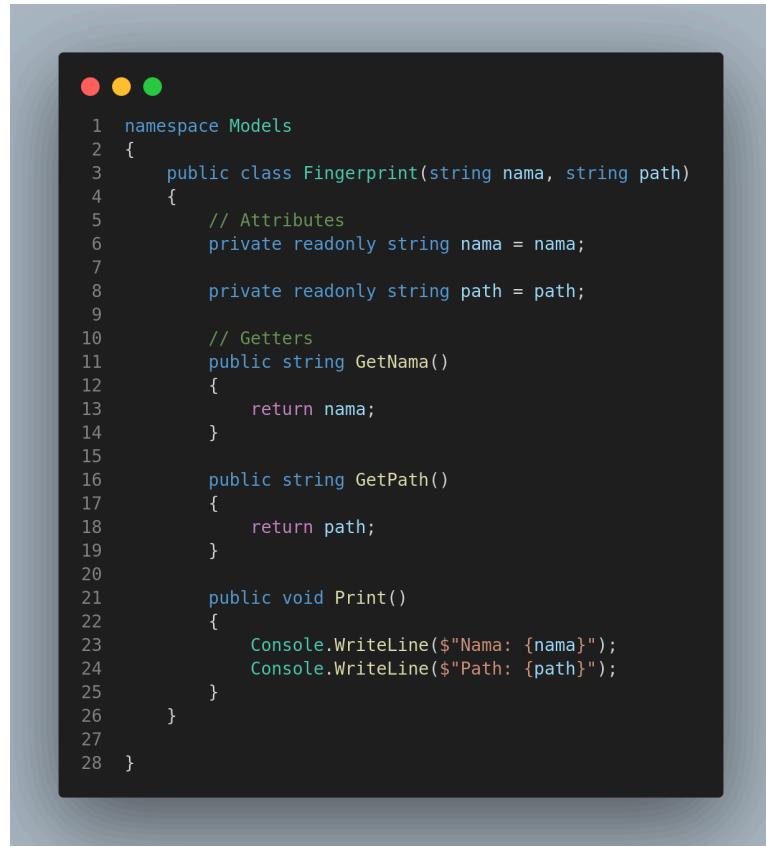


```
1 public static bool BmSolver(string text, string pattern)
2 {
3     // Get length of both strings
4     int n = text.Length;
5     int m = pattern.Length;
6
7     // Create bad character array
8     int[] last = new int[256];
9
10    // Fill bad character array
11    int i;
12    for (i = 0; i < 256; i++)
13    {
14        last[i] = -1;
15    }
16
17    for (i = 0; i < m; i++)
18    {
19        last[pattern[i]] = i;
20    }
21
22    // Start from the right
23    i = m - 1;
24    // Not found if pattern is longer than text
25    if (i > n - 1)
26    {
27        return false;
28    }
29
30    int j = m - 1;
31    do
32    {
33        // If the characters are the same
34        if (pattern[j] == text[i])
35        {
36            // End of pattern (found pattern)
37            if (j == 0)
38            {
39                return true;
40            }
41            // Move to the left
42            else
43            {
44                i--;
45                j--;
46            }
47        }
48        // If the characters are different
49        else
50        {
51            int k = last[text[i]];
52            i = i + m - Math.Min(j, 1 + k);
53            j = m - 1;
54        }
55    } while (i <= n - 1);
56
57    return false;
58 }
59 }
```

Gambar 4.1.2 Implementasi BM pada program

## 4.1.3 Struktur Data Fingerprint

Fingerprint digunakan sebagai suatu objek yang menggambarkan keadaan suatu baris pada database, sehingga kelas Fingerprint memuat nama dan pathnya.



```
1  namespace Models
2  {
3      public class Fingerprint(string nama, string path)
4      {
5          // Attributes
6          private readonly string nama = nama;
7
8          private readonly string path = path;
9
10         // Getters
11         public string GetNama()
12         {
13             return nama;
14         }
15
16         public string GetPath()
17         {
18             return path;
19         }
20
21         public void Print()
22         {
23             Console.WriteLine($"Nama: {nama}");
24             Console.WriteLine($"Path: {path}");
25         }
26     }
27 }
28 }
```

Gambar 4.1.3 Struktur data Fingerprint

#### 4.1.4 Struktur Data User

Kelas user digunakan sebagai objek user yang ada pada database, memuat atribut nik, nama, tempata lahir, tanggal lahir, jenis kelamin, golongan darah, alamat, agama, statusPerkawinan, pekerjaan, dan kewarganegaraan sehingga merepresentasikan suatu pengguna di dalam dunia nyata.

```
1 public class User(string nik, string nama,
2     string tempatLahir, DateTime tanggalLahir, string jenisKelamin, string golonganDarah,
3     string alamat, string agama, string statusPerkawinan, string pekerjaan, string kewarganegaraan)
4 {
5     // Attributes
6     private readonly string nik = nik;
7     private readonly string nama = nama;
8     private readonly string tempatLahir = tempatLahir;
9     private readonly DateTime tanggalLahir = tanggalLahir;
10    private readonly string jenisKelamin = jenisKelamin;
11    private readonly string golonganDarah = golonganDarah;
12    private readonly string alamat = alamat;
13    private readonly string agama = agama;
14    private readonly string statusPerkawinan = statusPerkawinan;
15    private readonly string pekerjaan = pekerjaan;
16    private readonly string kewarganegaraan = kewarganegaraan;
17
18    // Getters
19    public string GetNik()
20    {
21        return nik;
22    }
23
24    public string GetNama()
25    {
26        return nama;
27    }
28
29    public string GetTempatLahir()
30    {
31        return tempatLahir;
32    }
33
34    public DateTime GetTanggalLahir()
35    {
36        return tanggalLahir;
37    }
38
39    public string GetJenisKelamin()
40    {
41        return jenisKelamin;
42    }
43
44    public string GetGolonganDarah()
45    {
46        return golonganDarah;
47    }
48
49    public string GetAlamat()
50    {
51        return alamat;
52    }
53
54    public string GetAgama()
55    {
56        return agama;
57    }
58
59    public string GetStatusPerkawinan()
60    {
61        return statusPerkawinan;
62    }
63
64    public string GetPekerjaan()
65    {
66        return pekerjaan;
67    }
68
69    public string GetKewarganegaraan()
70    {
71        return kewarganegaraan;
72    }
73}
74 }
```

**Tabel 4.1.4** Implementasi Struktur Data User

#### 4.1.4 Struktur Data Db

Kelas Db pada program merupakan kelas yang menghubungkan local database dengan program,

```
+ references
public class Db
{
    + references
    private static MySqlConnection? connection;
    + reference
    private static readonly string connectionString = Environment.GetEnvironmentVariable("DB_CONNECTION_STRING") ?? throw new Exception("DB_CONNECTION_STRING is not set");

    + references
    public static MySqlConnection GetConnection() ->

    + references
    public static void CloseConnection() ->

    + reference
    public static void Migrate() ->

    + reference
    public static void LoadDump() ->

    + references
    public static void Seed() ->

}
```

Gambar 4.1.4.1 Kelas Db pada Program

Untuk mendapatkan koneksi antara local database dengan program, diperlukan suatu koneksi, hal tersebut diimplementasikan pada method GetConnection, serta CloseConnection apabila connection sudah tidak diperlukan.



```
1  public static MySqlConnection GetConnection()
2  {
3      if (connection != null && connection.State == System.Data.ConnectionState.Open)
4      {
5          return connection;
6      }
7
8      connection = new MySqlConnection(connectionString);
9      connection.Open();
10
11     return connection;
12 }
13
14 public static void CloseConnection()
15 {
16     if (connection != null)
17     {
18         connection.Close();
19         connection = null;
20     }
21 }
```

Gambar 4.1.4.2 Method GetConnection dan CloseConnection pada kelas Db

```
1 public static void Migrate()
2 {
3     // Create database if not exists & migrate schema from schema.sql
4     using MySqlConnection connection = GetConnection();
5     // Use transaction to rollback if error
6     MySqlTransaction transaction = connection.BeginTransaction();
7     MySqlCommand command = connection.CreateCommand();
8     command.Transaction = transaction;
9
10    // Read schema.sql
11    string schema = System.IO.File.ReadAllText("db/schema.sql");
12
13    // Execute schema.sql
14    try
15    {
16        // MIGRATE SCHEMA
17        command.CommandText = schema;
18        command.ExecuteNonQuery();
19
20        // COMMIT
21        transaction.Commit();
22    }
23    catch (Exception e)
24    {
25        Console.WriteLine(e.Message);
26        transaction.Rollback();
27    }
28    finally
29    {
30        CloseConnection();
31    }
32}
```

**Gambar 4.1.4.3** Method Migrate pada Kelas Db

```
1 public static void LoadDump()
2 {
3     // Create database if not exists & migrate schmea from schema.sql
4     using MySqlConnection connection = GetConnection();
5     // Use transaction to rollback if error
6     MySqlTransaction transaction = connection.BeginTransaction();
7     MySqlCommand command = connection.CreateCommand();
8     command.Transaction = transaction;
9
10    // Read schema.sql
11    string schema = System.IO.File.ReadAllText("db/seeded.sql");
12
13    // Execute schema.sql
14    try
15    {
16        command.CommandText = schema;
17        command.ExecuteNonQuery();
18        transaction.Commit();
19    }
20    catch (Exception e)
21    {
22        Console.WriteLine(e.Message);
23        transaction.Rollback();
24    }
25    finally
26    {
27        CloseConnection();
28    }
29 }
```

**Gambar 4.1.4.4** Method LoadDump pada kelas Db

```

1 public static void Seed()
2 {
3     // Generate
4     string testImagesPath = "../test/real/";
5 
6     // Initialize AES Encryption
7     LibAES aes = new();
8 
9     // Get all files in test/ folder
10    // Contains 6000 images where each person has 10 fingerprint images (from each fingers)
11    // Save relative path from src (which is ../test/real/<FILENAME>.BMP)
12    // Need to sort based on file name to make sure the order is correct
13    List<string> filesDirectories = new(Directory.GetFiles(testImagesPath).OrderBy(path => Convert.ToInt32(path.Split("../test/real/")[1].Split("_")[0])));
14    // Generate 600 names
15    List<string> names = new();
16    for (int i = 0; i < 600; i++)
17    {
18        names.Add(new Faker().Name.FullName());
19    }
20 
21    // Enums
22    List<string> golonganDarahEnum = ["A", "B", "AB", "O"];
23    List<string> agamaEnum = ["Islam", "Kristen", "Katolik", "Hindu", "Buddha", "Konfucius"];
24    List<string> statusPerkawinanEnum = ["Belum Menikah", "Menikah", "Cerai"];
25 
26    // Delete all data in database
27    using MySqlConnection connection = GetConnection();
28    MySqlCommand command = connection.CreateCommand();
29    MySqlTransaction transaction = connection.BeginTransaction();
30    command.Transaction = transaction;
31 
32    try
33    {
34        // Delete previous biodata
35        command.CommandText = "DELETE FROM biodata";
36        command.ExecuteNonQuery();
37 
38        // Delete previous sidik jari
39        command.CommandText = "DELETE FROM sidik_jari";
40        command.ExecuteNonQuery();
41 
42        // Insert new biodata
43        // Generate 600 biodata
44        for (int i = 0; i < names.Count; i++)
45        {
46            // Original data
47            // 50/50 chance data to be corrupted in biodata (alay name)
48            bool shouldAlay = new Random().Next(0, 2) == 0;
49            string nama = shouldAlay ? LibUtils.GetBahasaAlay(names[i]) : names[i];
50            string nik = (i + 1).ToString();
51            string tempatLahir = new Faker().Address.City();
52            string tanggalLahir = new Faker().Person.DateOfBirth.ToString("yyyy-MM-dd");
53            string jenisKelamin = new Random().Next(0, 2) == 0 ? "Laki-Laki" : "Perempuan";
54            string golonganDarah = golonganDarahEnum[new Random().Next(0, 4)];
55            string alamat = new Faker().Address.FullAddress();
56            string agama = agamaEnum[new Random().Next(0, 6)];
57            string statusPerkawinan = statusPerkawinanEnum[new Random().Next(0, 3)];
58            string pekerjaan = new Faker().Person.Company.Name;
59            string kewarganegaraan = new Faker().Address.CountryCode;
60 
61            // Encrypted data
62            string namaEncrypted = aes.Encrypt(nama).Replace("", @"\\");
63            string nikEncrypted = aes.Encrypt(nik).Replace("", @"\\");
64            string tempatLahirEncrypted = aes.Encrypt(tempatLahir).Replace("", @"\\");
65            string tanggalLahirEncrypted = aes.Encrypt(tanggalLahir).Replace("", @"\\");
66            string jenisKelaminEncrypted = aes.Encrypt(jenisKelamin).Replace("", @"\\");
67            string golonganDarahEncrypted = aes.Encrypt(golonganDarah).Replace("", @"\\");
68            string alamatEncrypted = aes.Encrypt(alamat).Replace("", @"\\");
69            string agamaEncrypted = aes.Encrypt(agama).Replace("", @"\\");
70            string statusPerkawinanEncrypted = aes.Encrypt(statusPerkawinan).Replace("", @"\\");
71            string pekerjaanEncrypted = aes.Encrypt(pekerjaan).Replace("", @"\\");
72            string kewarganegaraanEncrypted = aes.Encrypt(kewarganegaraan).Replace("", @"\\");
73 
74            // Insert data
75            command.CommandText = @"$#INSERT INTO biodata
76                (nik, nama, tempat_lahir, tanggal_lahir, jenis_kelamin, golongan_darah, alamat, agama, status_perkawinan, pekerjaan, kewarganegaraan)
77                VALUES
78                ('{nikEncrypted}', '{namaEncrypted}', '{tempatLahirEncrypted}', '{tanggalLahirEncrypted}', '{jenisKelaminEncrypted}', '{golonganDarahEncrypted}'
79                ,'{alamatEncrypted}', '{agamaEncrypted}', '{statusPerkawinanEncrypted}', '{pekerjaanEncrypted}', '{kewarganegaraanEncrypted}');
80            command.ExecuteNonQuery();
81        }
82 
83        // Generate 6000 fingerprints
84        for (int i = 0; i < filesDirectories.Count; i++)
85        {
86            // Original data
87            string nama = names[i / 10];
88            string path = filesDirectories[i];
89 
90            // Encrypt data
91            string encryptedNama = aes.Encrypt(nama).Replace("", @"\\");
92            string encryptedPath = aes.Encrypt(path).Replace("", @"\\");
93 
94            // Insert
95            command.CommandText = $"INSERT INTO sidik_jari (nama, berkas_citra) VALUES ('{encryptedNama}', '{encryptedPath}')";
96            command.ExecuteNonQuery();
97        }
98 
99        // Commit transaction
100        transaction.Commit();
101    }
102    catch (Exception e)
103    {
104        // Rollback transaction if error
105        Console.WriteLine(e.Message);
106        transaction.Rollback();
107    }
108    finally
109    {
110        CloseConnection();
111    }
112 }

```

**Gambar 4.1.4.5 Method Seed.**

## 4.1.5 Namespace lib

### 4.1.5.1 Struktur data ImageConverter

Struktur data ImageConverter digunakan untuk mengkonversi Image menjadi suatu image lain yang sudah dicrop, ataupun memuat utilitas dari suatu image, seperti konversi image menjadi ascii, mendapatkan titik tengah image.

```
10 references
public class ImageConverter
{
    /* Get image center size of size */
    0 references
    public static Image<Rgba32> GetCenteredImage(Image<Rgba32> image, int size) ...

    1 reference
    public static Image<Rgba32> GetCenteredImage(Image<Rgba32> image, int size, int top) ...

    /* Convert image to a binary string representation */
    0 references
    public static string ConvertToBinary(Image<Rgba32> image) ...

    /* Convert image to a ASCII 8 bits representation */
    4 references
    public static string ConvertToAscii8Bits(Image<Rgba32> image) ...

        /* Convert image to a ASCII 8 bits representation */
        0 references
        public static char[,] ConvertToAscii8BitsArray(Image<Rgba32> image) ...

    4 references
    public static Image<Rgba32> OmitWhiteSpace (Image<Rgba32> image) [...] ...

    0 references
    public static Image<Rgba32>[] GetThreeCenteredImage(Image<Rgba32> image, int size) ...

    2 references
    public static string[] GetCenteredArrays(string originalArray) { ... }
```

Gambar 4.1.5.1 Struktur data ImageConverter yang diimplementasi

#### 4.1.5.2 Implementasi Algoritma AES

```
9 references
public class AES
{
    // Private key state
    3 references
    private readonly byte[] key;

    // S-Box
    1 reference
    private readonly byte[,] sBox = new byte[16, 16] { ... };

    // Inverse S-Box
    1 reference
    private readonly byte[,] invSBox = new byte[16, 16] { ... };

    // Constructor
    4 references
    public AES() ...

    // Encrypt method
    14 references
    public string Encrypt(string plainText) ...

    14 references
    public string Decrypt(string cipherText) ...

    1 reference
    private byte[] ApplyPKCS7Padding(byte[] inputBytes) ...

    1 reference
    private byte[] RemovePKCS7Padding(byte[] inputBytes) ...

    2 references
    private byte[,] KeyExpansion(byte[] key) ...

    2 references
    private void SubBytes(byte[,] state) ...

    2 references
    private void ShiftRows(byte[,] state) ...

    1 reference
    private void MixColumns(byte[,] state) ...

    6 references
    private void AddRoundKey(byte[,] state, byte[,] roundKey, int round) ...

    2 references
    private void InvSubBytes(byte[,] state) ...

    2 references
    private void InvShiftRows(byte[,] state) ...

    1 reference
    private void InvMixColumns(byte[,] state) ...

    24 references
    private byte Gmul(byte a, byte b) ...
}
```

Gambar 4.1.5.2 Method dan Atribut pada Algoritma AES

Berikut adalah langkah-langkah dari algoritma AES yang digunakan sehingga AES memerlukan method dan atribut seperti pada gambar.

**1. Inisialisasi Kunci:**

Kunci enkripsi diambil dari variabel lingkungan PRIVATE\_KEY yang panjangnya harus 32 karakter (256 bit).

**2. Konversi Teks ke Byte**

Teks plaintext yang akan dienkripsi dikonversi menjadi array byte menggunakan Encoding.UTF8.GetBytes().

**3. Padding PKCS7:**

Jika panjang plaintext bukan kelipatan dari 16 byte, maka padding PKCS7 diterapkan untuk memastikan panjang data adalah kelipatan dari 16 byte. Fungsi ApplyPKCS7Padding menambahkan padding.

**4. Pembagian Blok:**

Data yang sudah dipadding kemudian dibagi menjadi blok-blok 16 byte.

**5. Ekspansi Kunci:**

Kunci utama diekspansi menjadi kunci round menggunakan fungsi KeyExpansion.

**6. Enkripsi Setiap Blok:**

Setiap blok dienkripsi secara terpisah:

a. **Inisialisasi State:**

i. Data 16 byte dari blok diinisialisasi ke dalam array 4x4 byte (state).

b. **AddRoundKey Awal:**

i. Kunci round awal ditambahkan ke state.

c. **Ronde Enkripsi:**

i. 13 ronde enkripsi yang melibatkan operasi SubBytes, ShiftRows, MixColumns, dan AddRoundKey.

ii. Pada ronde ke-14 (terakhir), hanya operasi SubBytes, ShiftRows, dan AddRoundKey dilakukan tanpa MixColumns.

**7. Penggabungan Hasil:**

Blok-blok terenkripsi digabungkan kembali menjadi array byte hasil enkripsi.

## 8. Konversi ke Base64:

Hasil enkripsi dikonversi ke dalam format Base64 untuk memudahkan penyimpanan dan pengiriman data teks.

### 4.1.6 Program Utama

#### 4.1.6.1 Hubungan GUI dengan Algoritma



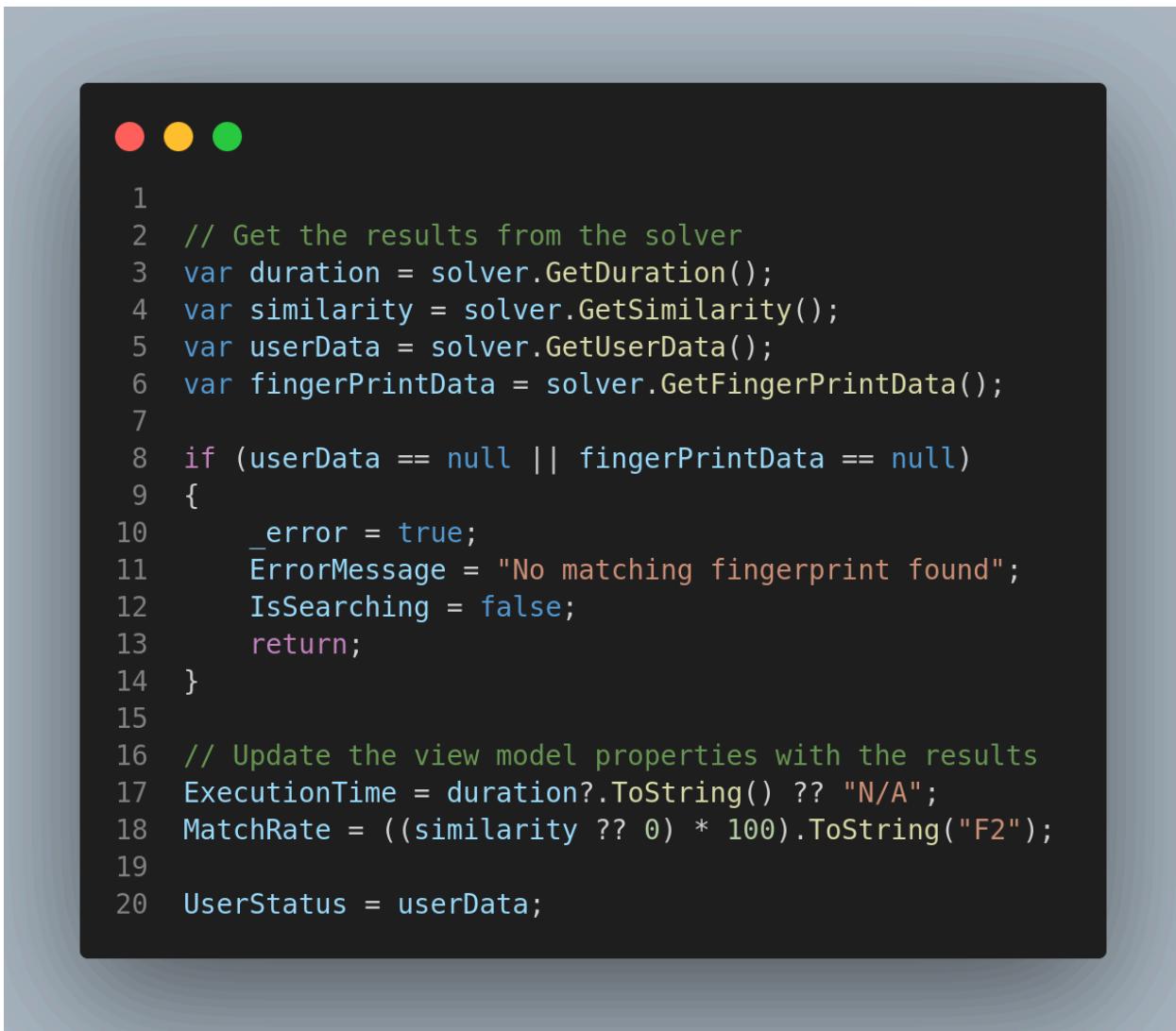
Gambar 4.1.6.1.1 Gambar GUI yang dibuat.

Search command yang dibuat pada gui merupakan button yang apabila diklik memanggil fungsi command yang didalam fungsi tersebut membuat objek solver

```
1 Solver solver = new (filePath, IsKMPChecked);  
2  
3 // Perform the solve operation asynchronously  
4 await Task.Run(() => solver.Solve());
```

Gambar 4.1.6.1.2 Kode program yang memanggil solver.

Yang kemudian menampilkan datanya pada GUI melalui getter dari solver.



The screenshot shows a dark-themed code editor window with three colored window controls (red, yellow, green) at the top. The code is written in C# and performs the following steps:

- Get results from the solver.
- Check if userData and fingerPrintData are null.
- If they are null, set \_error to true, ErrorMessage to "No matching fingerprint found", IsSearching to false, and return.
- Update view model properties with the results: ExecutionTime, MatchRate, and UserStatus.

```
1 // Get the results from the solver
2 var duration = solver.GetDuration();
3 var similarity = solver.GetSimilarity();
4 var userData = solver.GetUserData();
5 var fingerPrintData = solver.GetFingerPrintData();
6
7
8 if (userData == null || fingerPrintData == null)
9 {
10     _error = true;
11     ErrorMessage = "No matching fingerprint found";
12     IsSearching = false;
13     return;
14 }
15
16 // Update the view model properties with the results
17 ExecutionTime = duration?.ToString() ?? "N/A";
18 MatchRate = ((similarity ?? 0) * 100).ToString("F2");
19
20 UserStatus = userData;
```

Gambar 4.1.6.1.3 Update state setelah searching.

#### 4.1.6.2 Method solve pada Solver yang diimplementasikan

```
1 // Start calculation duration
2 var watch = System.Diagnostics.Stopwatch.StartNew();
3
4 // Get all fingerprints
5 List<Models.Fingerprint> allFingerprints = Controllers.Fingerprint.GetFingerprints();
6
7 // Get input image
8 Image<Rgba32> inputImage = Image.Load<Rgba32>(inputImagePath);
9
10
11 inputImage = Lib.ImageConverter.OmitWhiteSpace(inputImage);
12
13
14
15 string inputImageAsciiString = Lib.ImageConverter.ConvertToAscii8Bits(inputImage);
16 string[] inputImageCropped = Lib.ImageConverter.GetCenteredArrays(inputImageAsciiString);
17
18 // Iterate through all fingerprints
19 bool isMatch = false;
20 Models.Fingerprint? bestFingerprint = null;
21 double bestSimilarity = 0;
```

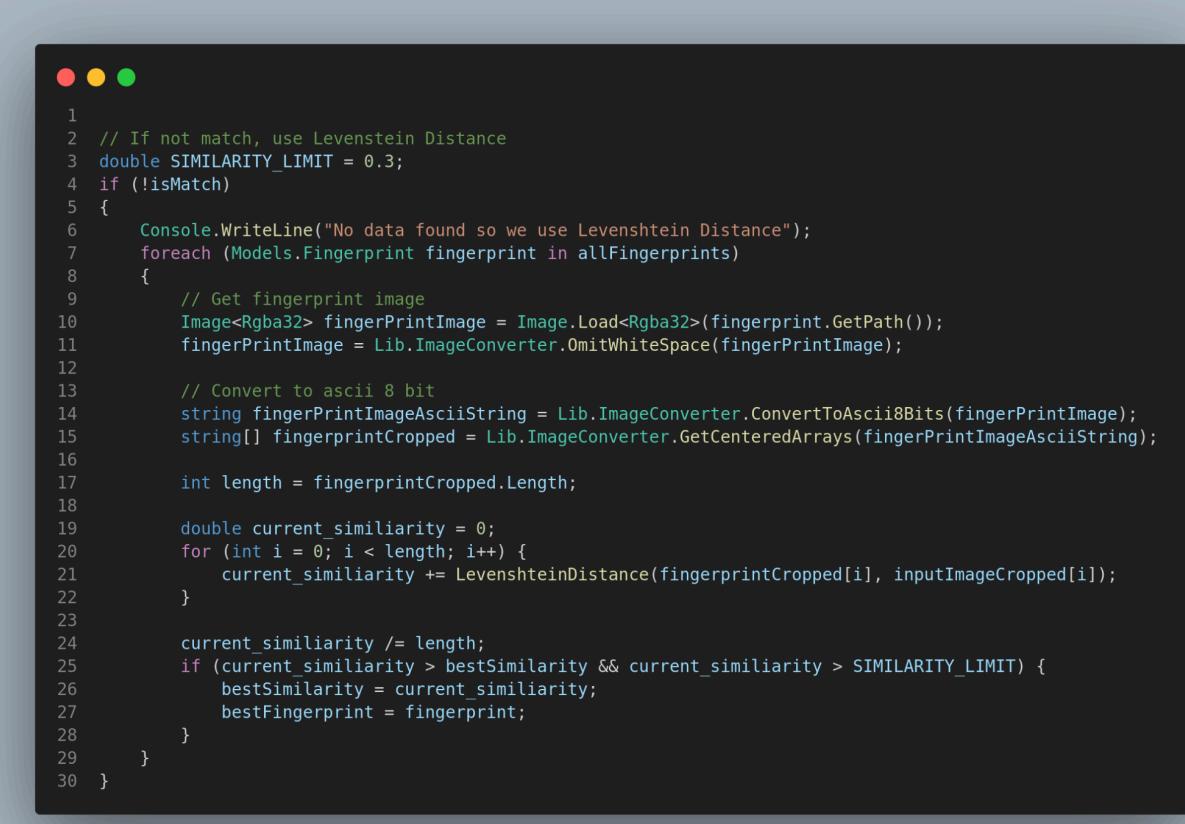
**Gambar 4.1.6.2.1** method solve bagian 1

Method solve dimulai dengan menyimpan time pada method solve dipanggil, kemudian mengambil dari database semua fingerprint yang tersedia, kemudian menyiapkan variabel-variabel yang dibutuhkan untuk melakukan string matching. Kemudian dilakukan algoritma KMP atau BM.

```
1
2 // Solve for KMP
3 if (isKmp)
4 {
5     foreach (Models.Fingerprint fingerprint in allFingerprints)
6     {
7         foreach (string pattern in inputImageCropped)
8         {
9             // Get fingerprint image
10            Image<Rgba32> fingerPrintImage = Image.Load<Rgba32>(fingerprint.GetPath());
11            fingerPrintImage = Lib.ImageConverter.OmitWhiteSpace(fingerPrintImage);
12
13            // Convert to ascii 8 bit
14            string fingerPrintImageAsciiString = Lib.ImageConverter.ConvertToAscii8Bits(fingerPrintImage);
15
16            // Input string as pattern
17            // Finger print string as text
18            isMatch = KmpSolver(fingerPrintImageAsciiString, pattern);
19
20            if (isMatch)
21            {
22                // Save fingerprint (contains name & path)
23                double current_similarity = LevenshteinDistance(fingerPrintImageAsciiString, inputImageAsciiString);
24                if (current_similarity > bestSimilarity && current_similarity > 0.5) {
25                    bestFingerprint = fingerprint;
26                    bestSimilarity = current_similarity;
27                }
28            }
29        }
30
31        if (isMatch) {
32            break;
33        }
34    }
35 }
36 // Solve for BM
37 else
38 {
39     foreach (Models.Fingerprint fingerprint in allFingerprints)
40     {
41
42         // Get fingerprint image
43         Image<Rgba32> fingerPrintImage = Image.Load<Rgba32>(fingerprint.GetPath());
44         fingerPrintImage = Lib.ImageConverter.OmitWhiteSpace(fingerPrintImage);
45
46         // Convert to ascii 8 bit
47         string fingerPrintImageAsciiString = Lib.ImageConverter.ConvertToAscii8Bits(fingerPrintImage);
48         int count = 0;
49         foreach (string pattern in inputImageCropped)
50         {
51             count++;
52             isMatch = BmSolver(fingerPrintImageAsciiString, pattern);
53
54
55             if (isMatch)
56             {
57                 // print pattern
58                 double current_similarity = LevenshteinDistance(fingerPrintImageAsciiString, inputImageAsciiString);
59                 if (current_similarity > bestSimilarity && current_similarity > 0.5) {
60                     bestFingerprint = fingerprint;
61                     bestSimilarity = current_similarity;
62                 }
63             }
64         }
65
66         if (isMatch) {
67             break;
68         }
69     }
70 }
```

**Gambar 4.1.6.2.2** method solve bagian 2

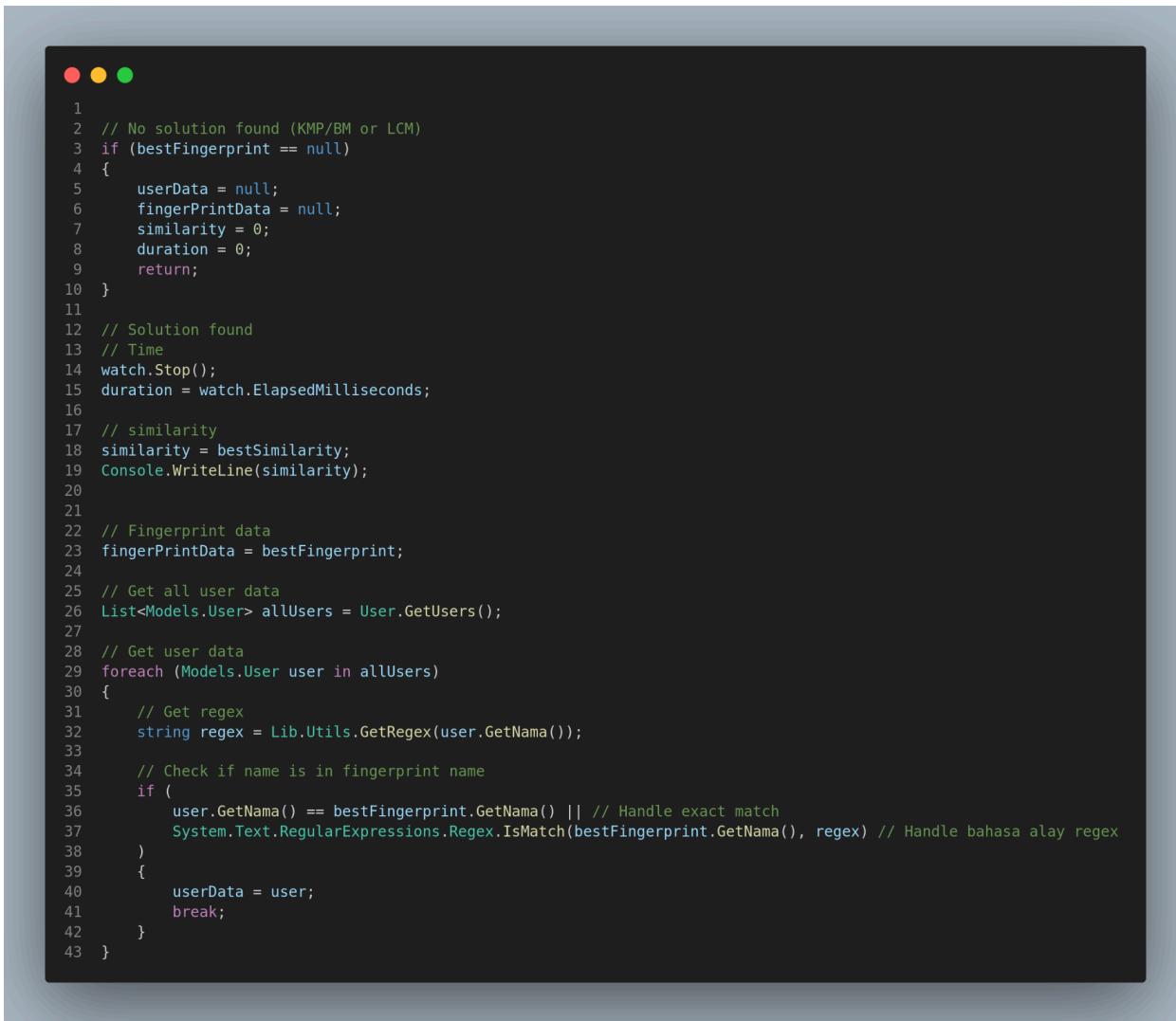
Apabila Tidak ditemukan, lakukan perbandingan antara 5 pattern pada input image dengan 5 pattern pada gambar finger print, kemudian lakukan levenshtein distance untuk mendapatkan similiarity antara 2 gambar. Kemudian ambil fingerprint dengan similiarity tertinggi.



```
1 // If not match, use Levenshtein Distance
2 double SIMILARITY_LIMIT = 0.3;
3 if (!isMatch)
4 {
5     Console.WriteLine("No data found so we use Levenshtein Distance");
6     foreach (Models.Fingerprint fingerprint in allFingerprints)
7     {
8         // Get fingerprint image
9         Image<Rgba32> fingerPrintImage = Image.Load<Rgba32>(fingerprint.GetPath());
10        fingerPrintImage = Lib.ImageConverter.OmitWhiteSpace(fingerPrintImage);
11
12        // Convert to ascii 8 bit
13        string fingerPrintImageAsciiString = Lib.ImageConverter.ConvertToAscii8Bits(fingerPrintImage);
14        string[] fingerPrintImageCropped = Lib.ImageConverter.GetCenteredArrays(fingerPrintImageAsciiString);
15
16        int length = fingerPrintImageCropped.Length;
17
18        double current_similiarity = 0;
19        for (int i = 0; i < length; i++) {
20            current_similiarity += LevenshteinDistance(fingerPrintImageCropped[i], inputImageCropped[i]);
21        }
22
23        current_similiarity /= length;
24        if (current_similiarity > bestSimilarity && current_similiarity > SIMILARITY_LIMIT) {
25            bestSimilarity = current_similiarity;
26            bestFingerprint = fingerPrintImage;
27        }
28    }
29 }
30 }
```

**Gambar 4.1.6.2.3** method solve bagian 3

Setelah itu siapkan attribut-attribut dari solver agar dapat diakses melalui getter dari luar objek.



```
1 // No solution found (KMP/BM or LCM)
2 if (bestFingerprint == null)
3 {
4     userData = null;
5     fingerPrintData = null;
6     similarity = 0;
7     duration = 0;
8     return;
9 }
10 }
11 // Solution found
12 // Time
13 watch.Stop();
14 duration = watch.ElapsedMilliseconds;
15
16 // similarity
17 similarity = bestSimilarity;
18 Console.WriteLine(similarity);
19
20
21 // Fingerprint data
22 fingerPrintData = bestFingerprint;
23
24
25 // Get all user data
26 List<Models.User> allUsers = User.GetUsers();
27
28 // Get user data
29 foreach (Models.User user in allUsers)
30 {
31     // Get regex
32     string regex = Lib.Utils.GetRegex(user.GetNama());
33
34     // Check if name is in fingerprint name
35     if (
36         user.GetNama() == bestFingerprint.GetNama() || // Handle exact match
37         System.Text.RegularExpressions.Regex.IsMatch(bestFingerprint.GetNama(), regex) // Handle bahasa alay regex
38     )
39     {
40         userData = user;
41         break;
42     }
43 }
```

**Gambar 4.1.6.2.4** method solve bagian 4

## 4.2. Penggunaan Program

### 4.2.1 Setup Database MySQL

1. Buka MySQL dengan user root lalu buat database bernama "tubes3\_stima".

```
CREATE DATABASE tubes3_stima;
```

2. Buat user dengan username "college" dan password "12345" di local database.

```
CREATE USER 'college'@'localhost' IDENTIFIED BY '12345';
```

3. Berikan semua privilege pada user tersebut pada database tubes3\_stima

```
GRANT ALL PRIVILEGES ON tubes3_stima.* TO 'college'@'localhost';
```

4. Load hasil seeding (dump pada src/db/seeded.sql)

```
chmod +x load-dump.sh
```

```
./load-dump.sh
```

## 4.2.2 Menjalankan Program

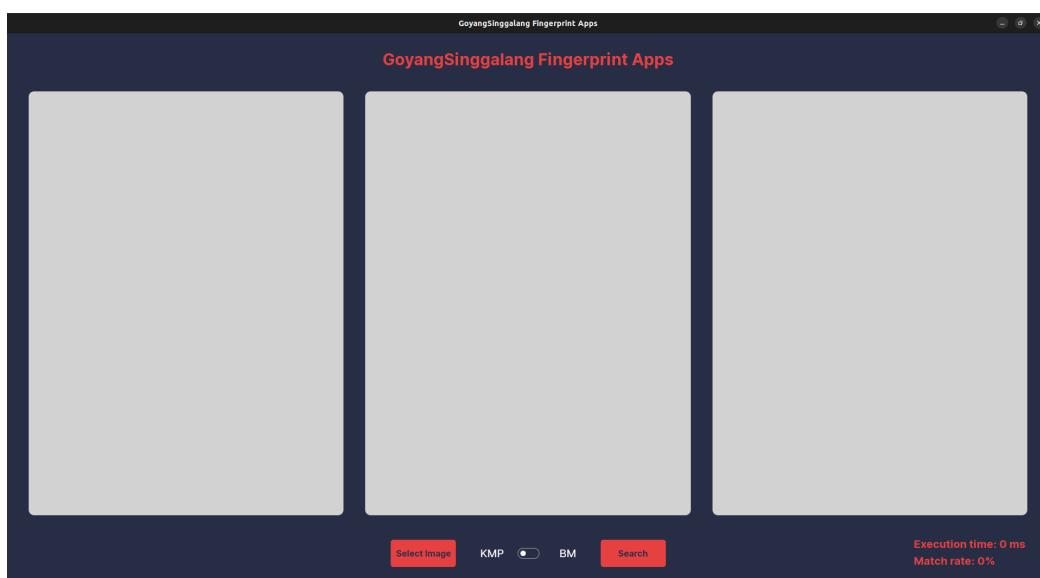
1. Jalankan script shell gui.sh

```
chmod +x gui.sh
```

```
./gui.sh
```

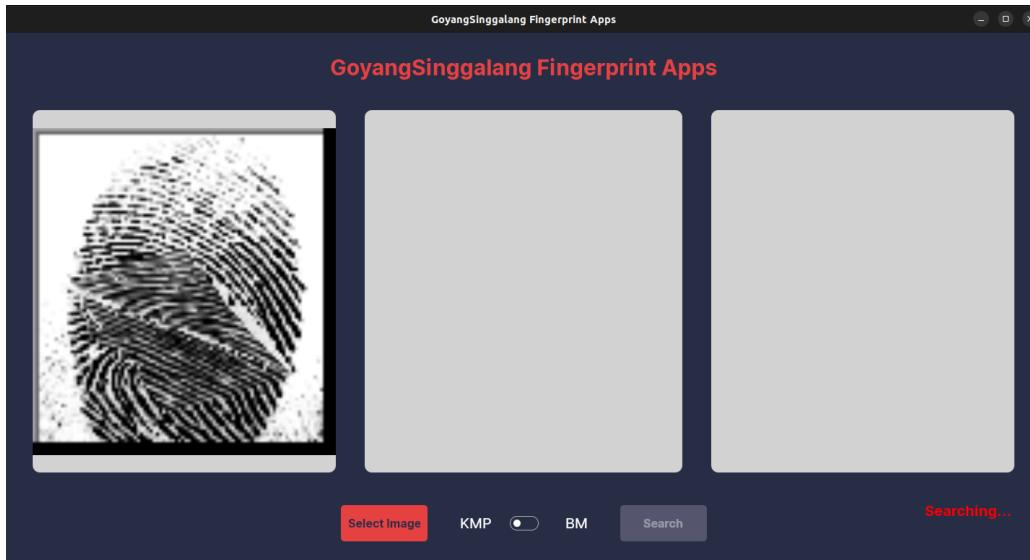
## 4.2.3 Menggunakan Program

1. Masukkan input gambar



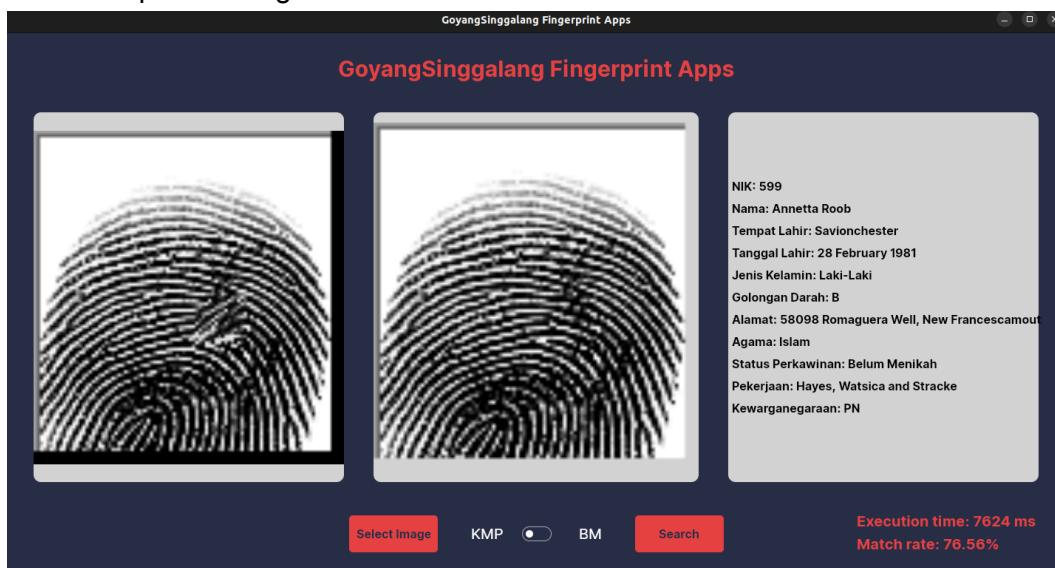
**Gambar 4.2.1** Tampilan hasil ditemukan

2. Tunggu pencarian gambar



**Gambar 4.2.2** Tampilan hasil ditemukan

3. Lihat hasil dari pencarian gambar

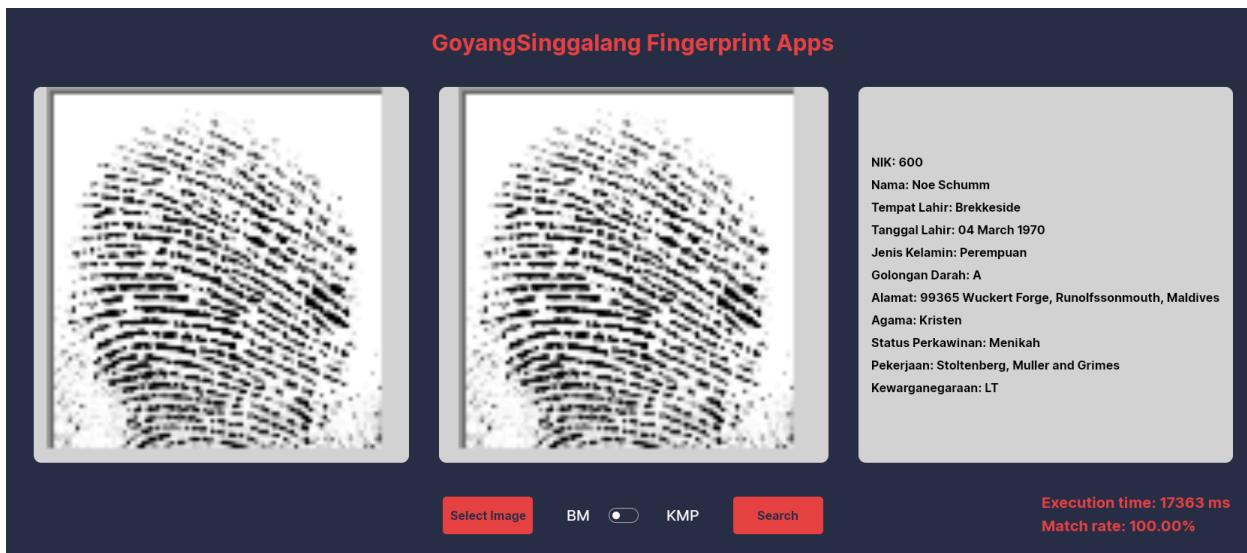


**Gambar 4.2.3** Tampilan hasil ditemukan

## 4.3. Uji Coba Program

### 4.3.1 Uji Coba Program

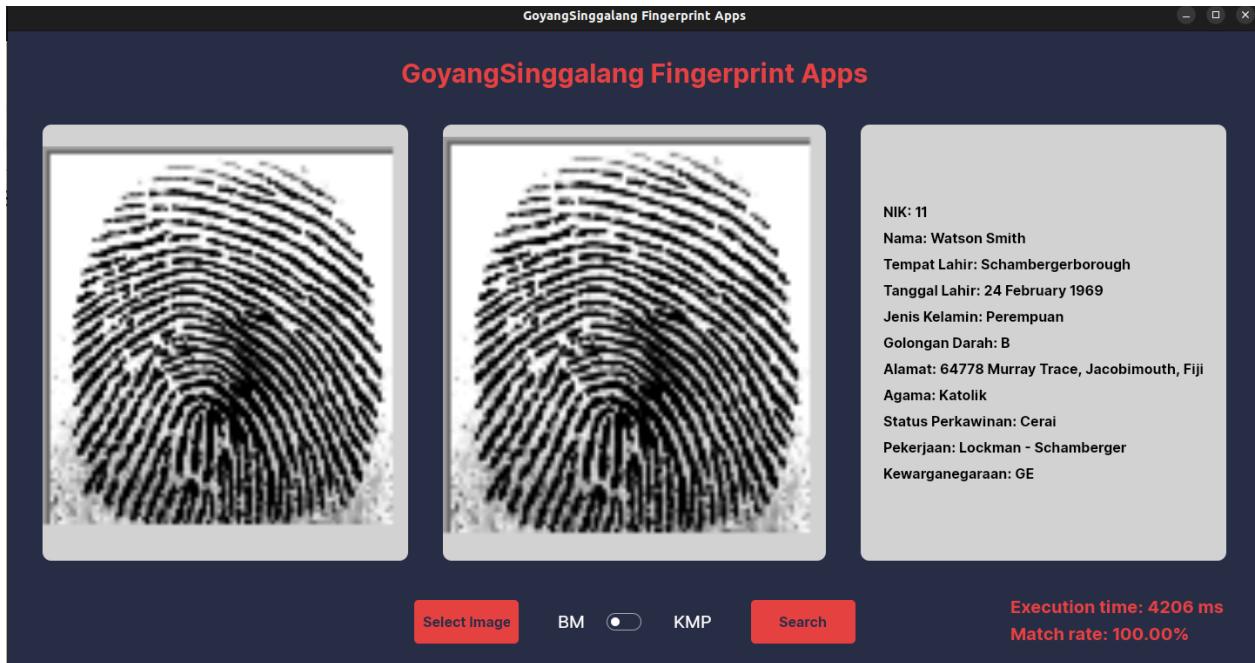
#### 4.3.1.1 Fingerprint Real



Gambar 4.3.1.1 Hasil test case 1 fingerprint real (ketemu match dan correct) dengan algoritma BM



Gambar 4.3.1.2 Hasil test case 2 fingerprint real (ketemu match dan correct) dengan algoritma KMP

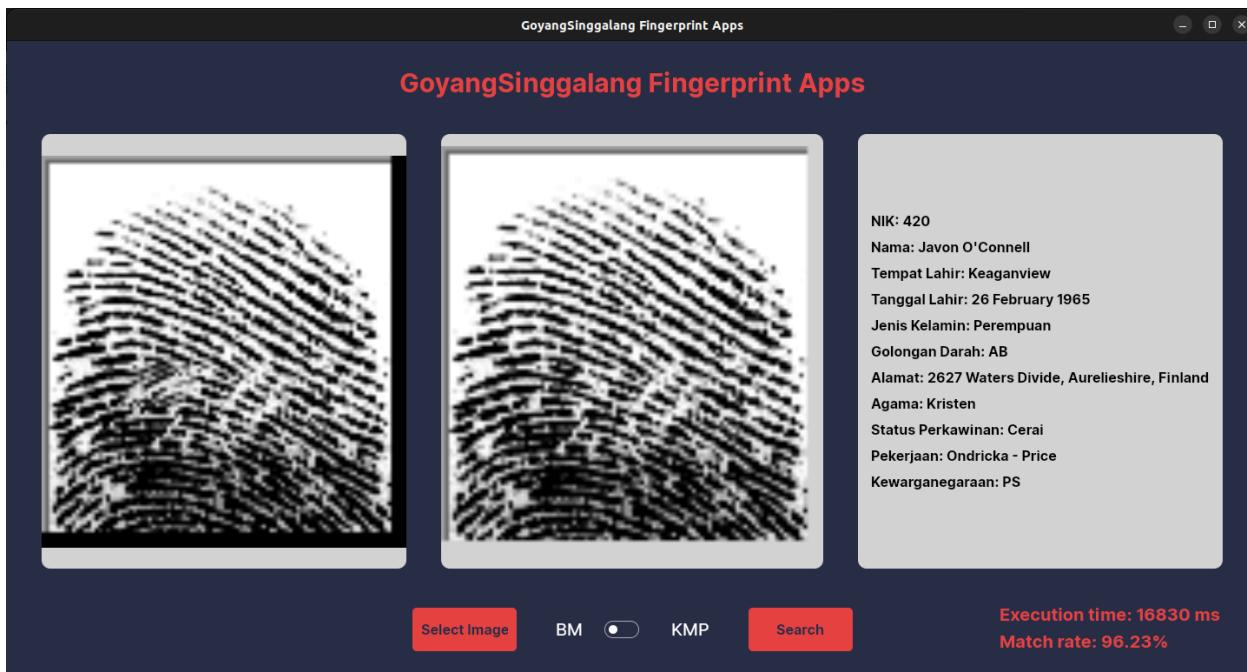


**Gambar 4.3.1.1.3** Hasil test case 3 fingerprint real (ketemu match dan correct) dengan algoritma BM

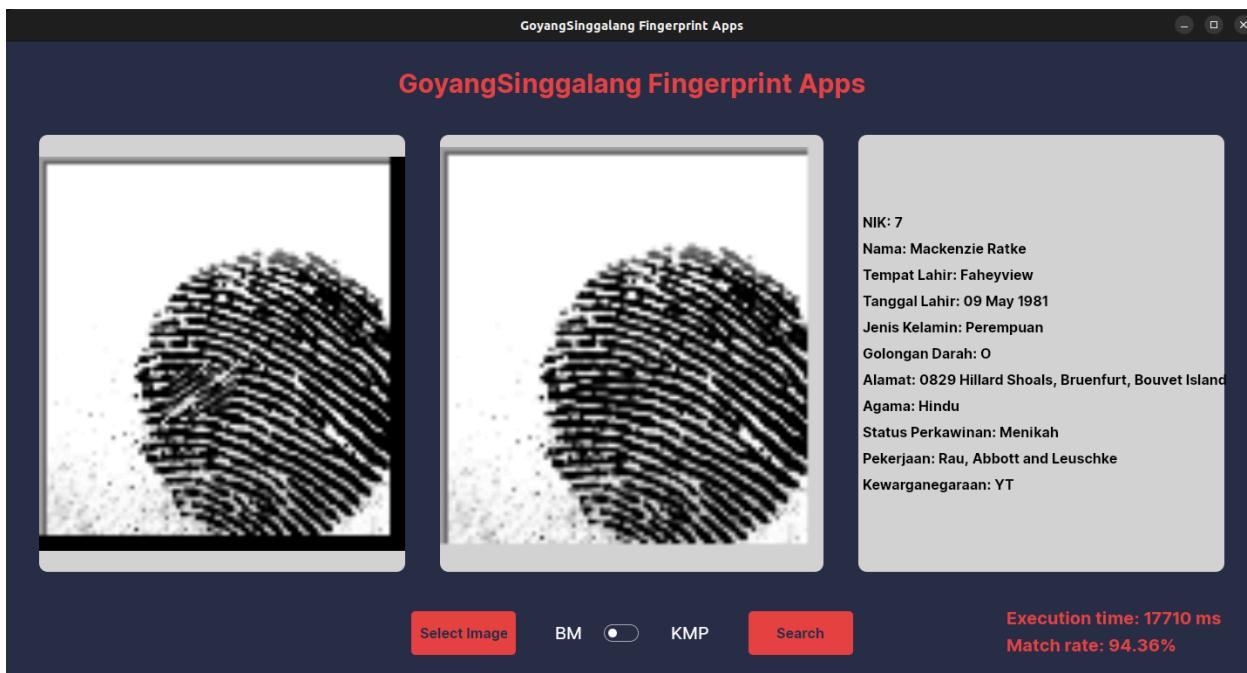


**Gambar 4.3.1.1.4** Hasil test case 4 fingerprint real (ketemu match dan correct) dengan algoritma BM

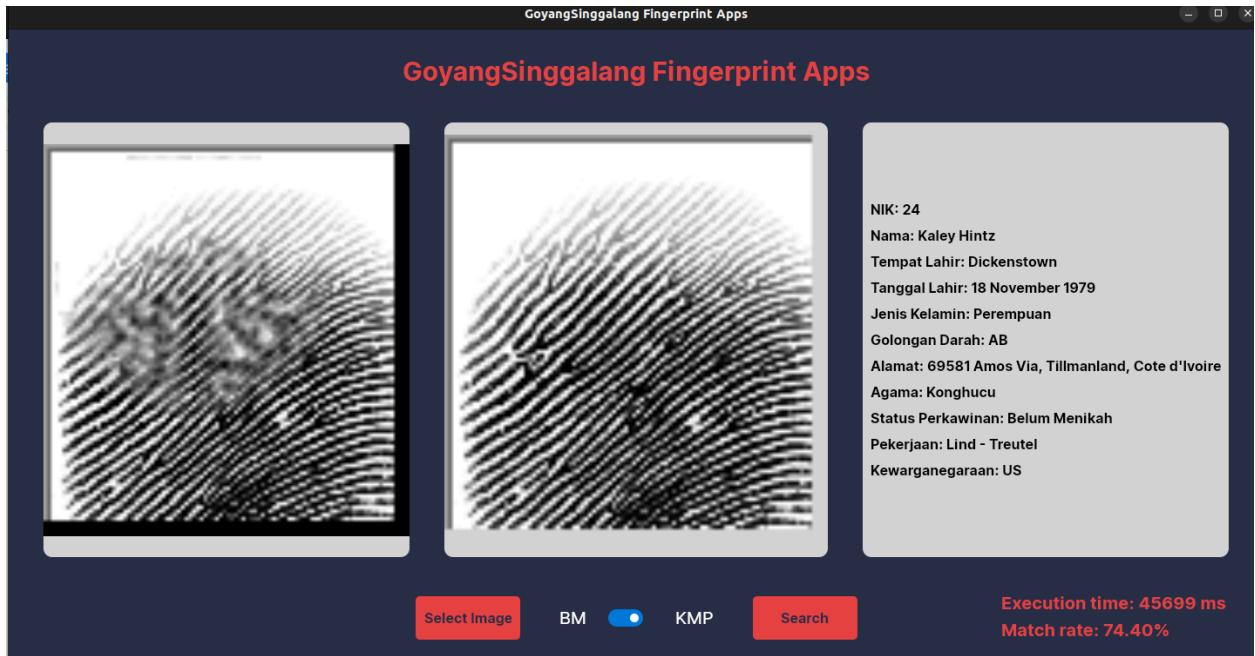
#### 4.3.1.2 Fingerprint Altered Easy



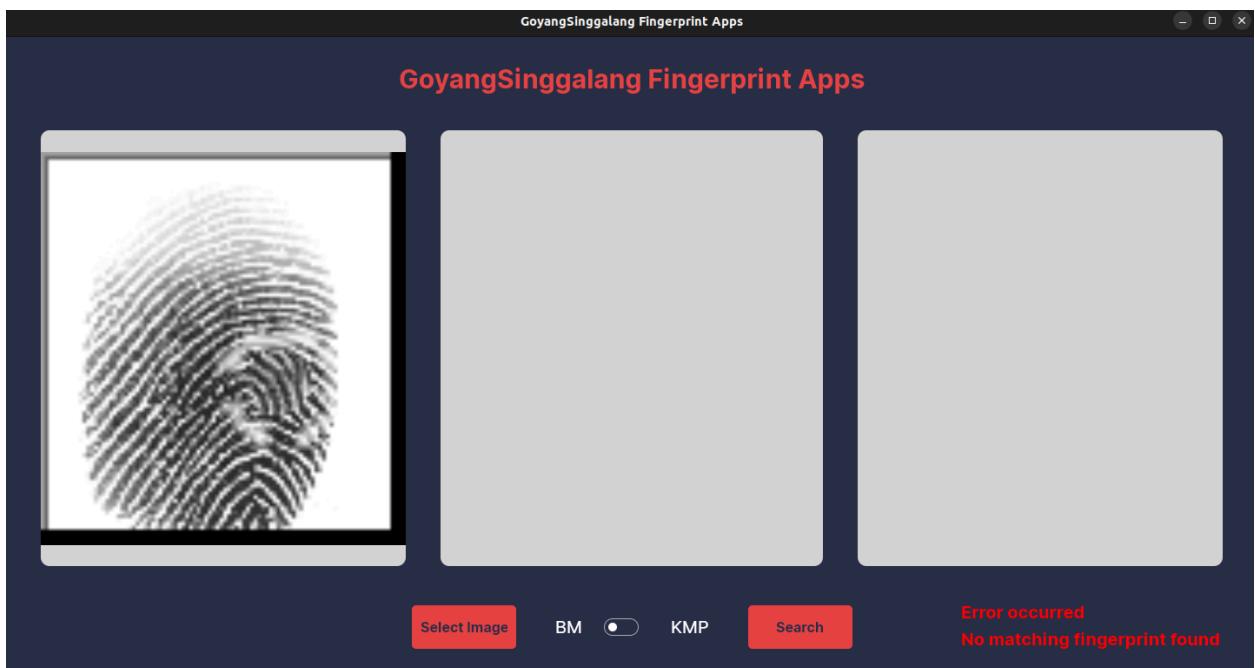
Gambar 4.3.1.2.1 Hasil test case 1 fingerprint altered easy (ketemu match dan correct) dengan algoritma BM



Gambar 4.3.1.2.2 Hasil test case 2 fingerprint altered easy (ketemu match dan correct) dengan algoritma BM

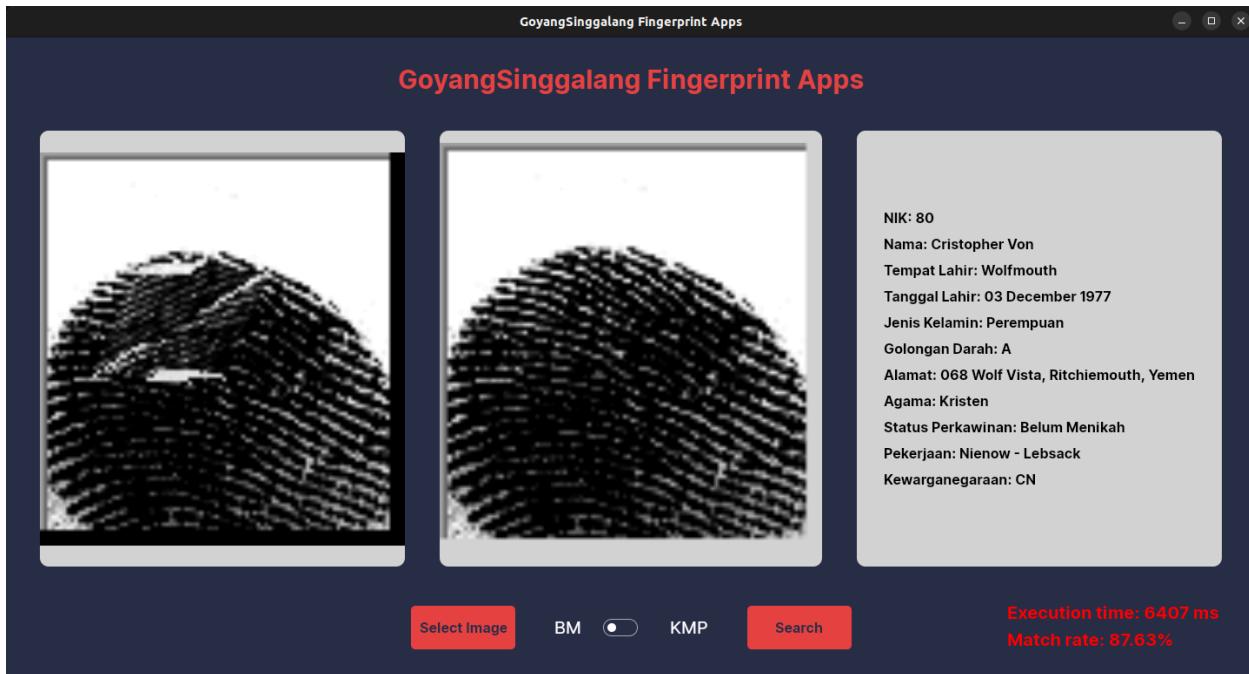


Gambar 4.3.1.2.3 Hasil test case 3 fingerprint altered easy (ketemu match dan correct) dengan algoritma KMP

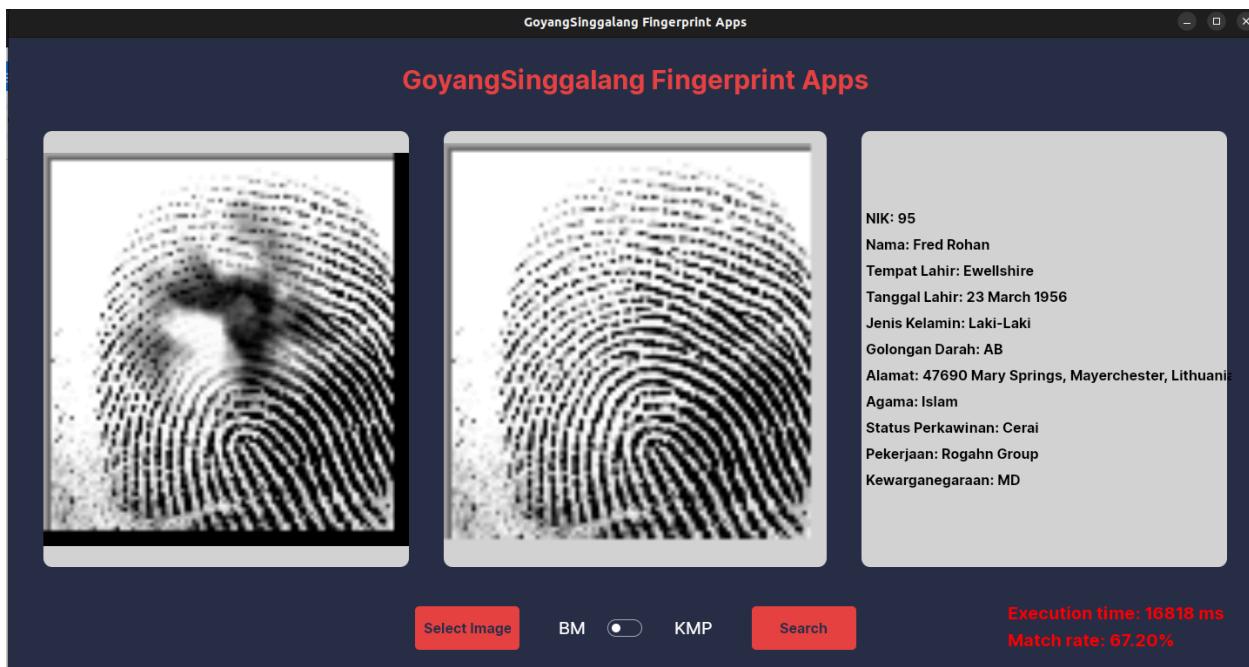


Gambar 4.3.1.2.4 Hasil test case 4 fingerprint altered easy tidak ketemu

#### 4.3.1.3 Fingerprint Altered Medium



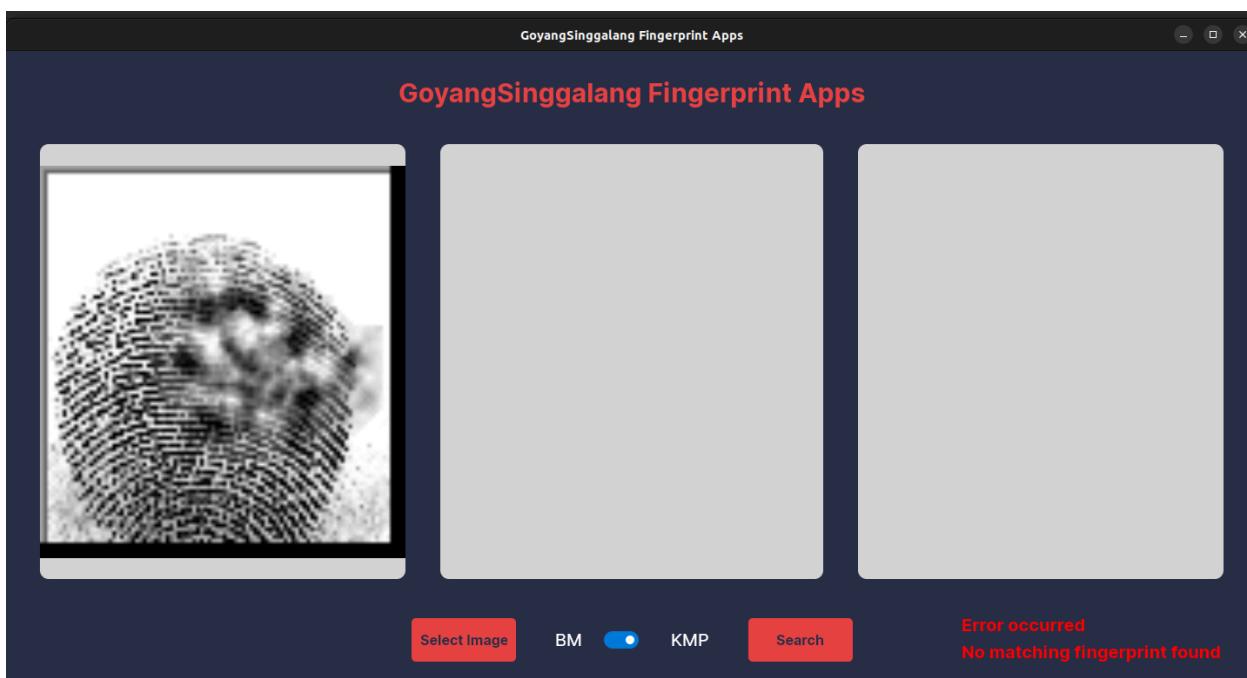
Gambar 4.3.1.3.1 Hasil test case 1 fingerprint altered medium (ketemu match dan correct) dengan algoritma BM



Gambar 4.3.1.3.2 Hasil test case 2 fingerprint altered medium (ketemu match dan correct) dengan algoritma BM

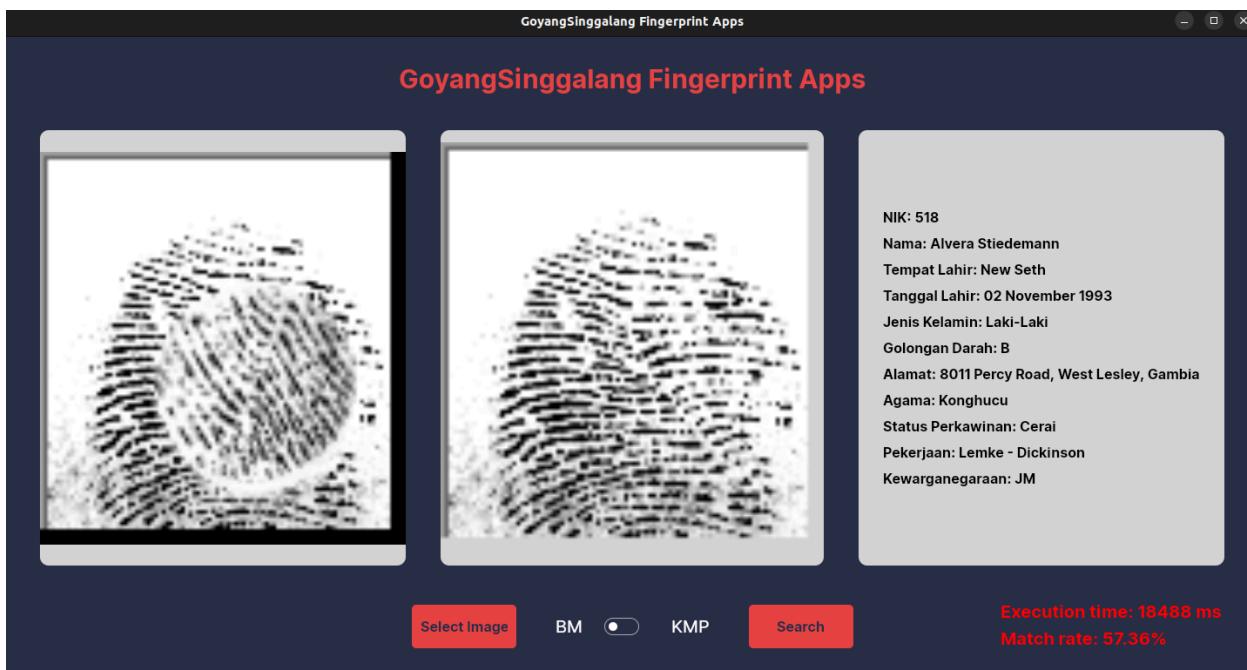


**Gambar 4.3.1.3.3** Hasil test case 3 fingerprint altered medium (ketemu match dan correct) dengan algoritma BM

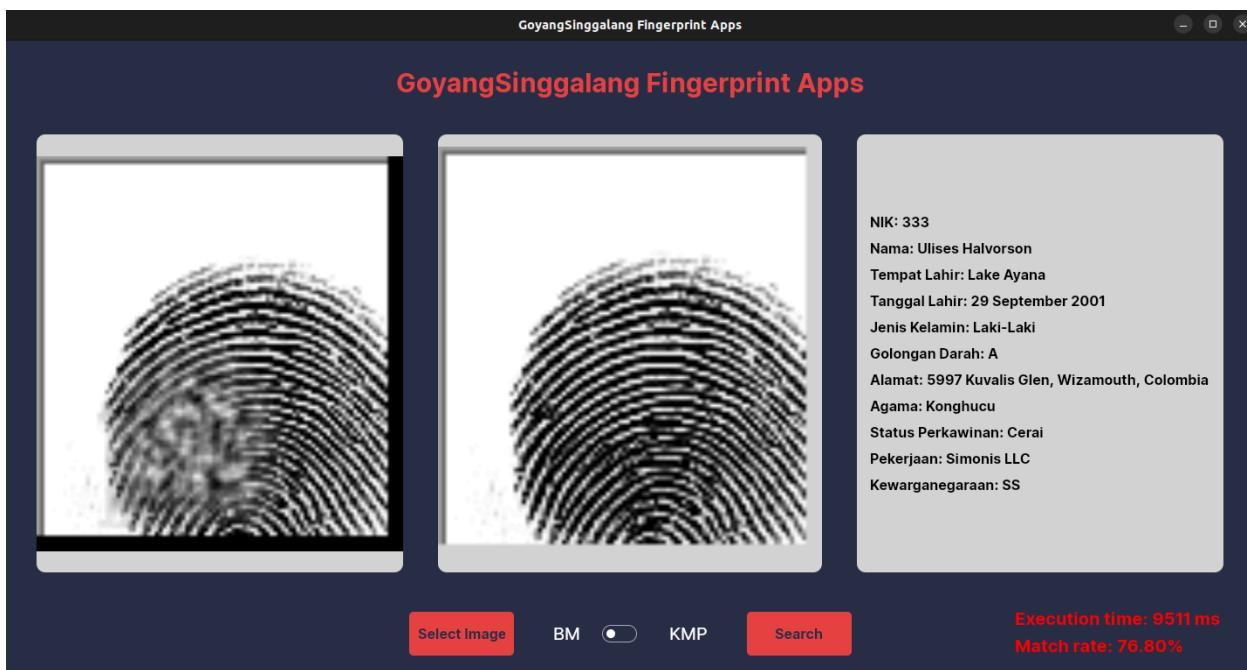


**Gambar 4.3.1.3.4** Hasil test case 4 fingerprint altered medium (tidak ketemu) dengan algoritma BM

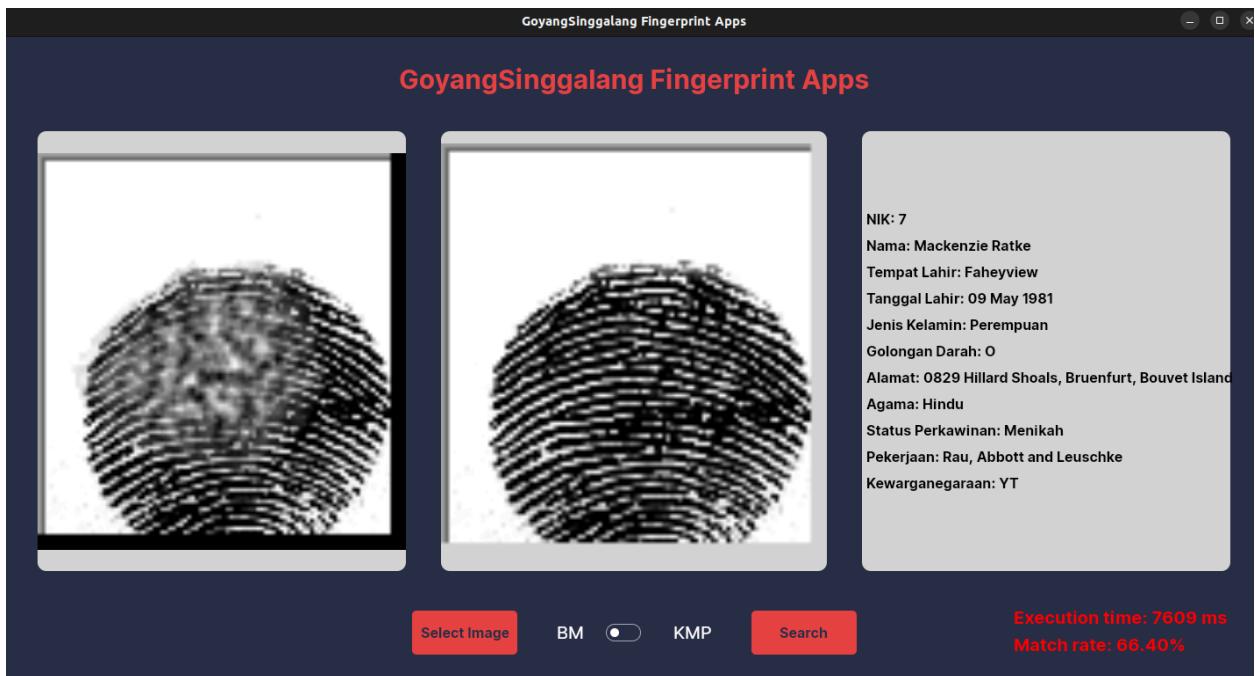
#### 4.3.1.4 Fingerprint Altered Hard



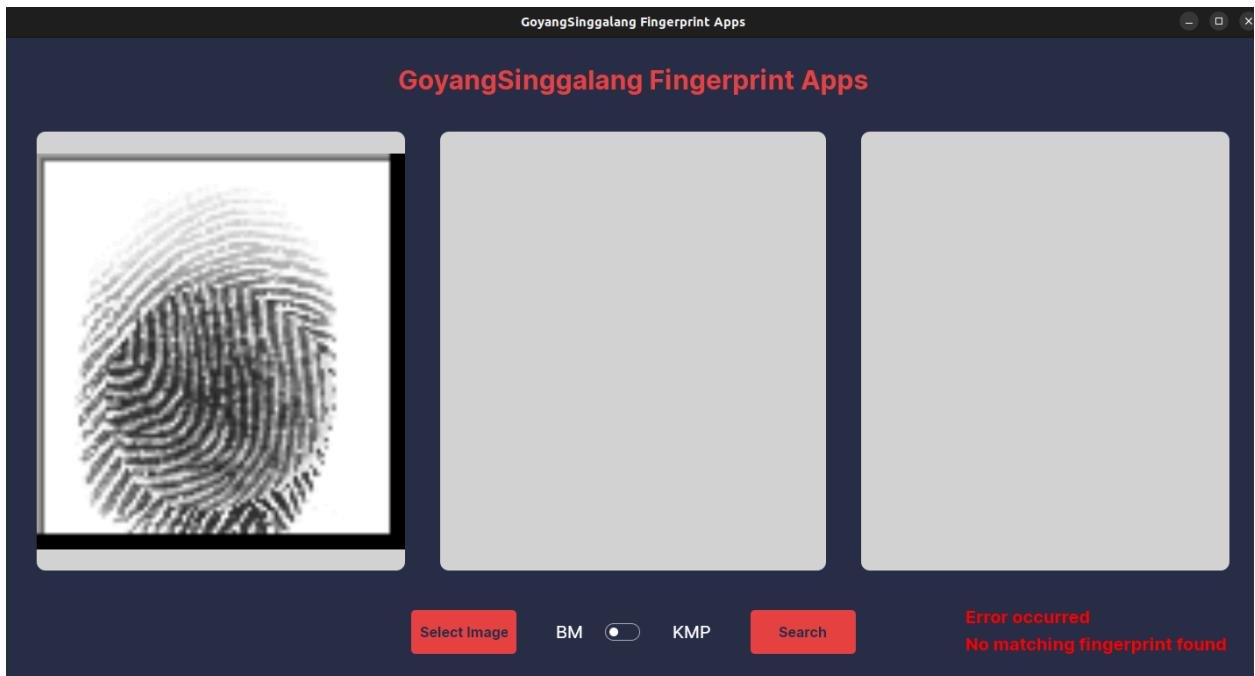
Gambar 4.3.1.4.1 Hasil test case 1 fingerprint altered hard (ketemu match dan correct) dengan algoritma BM



Gambar 4.3.1.4.2 Hasil test case 2 fingerprint altered hard (ketemu match dan correct) dengan algoritma BM



Gambar 4.3.1.4.3 Hasil test case 3 fingerprint altered hard (ketemu match dan correct) dengan algoritma BM



Gambar 4.3.1.4.4 Hasil test case 4 fingerprint altered hard (tidak ketemu) dengan algoritma BM

#### 4.3.2 Uji Coba Stress

Sourcecode uji coba stress dapat dilihat pada

[https://github.com/randyver/Tubes3\\_GoyangSinggalang/blob/main/src/Cli.cs#L134](https://github.com/randyver/Tubes3_GoyangSinggalang/blob/main/src/Cli.cs#L134).

Berikut adalah hasil dari stress test tiga jenis gambar (*altered easy*, *altered medium*, *altered hard*).

Match & CORRECT: 166/184  
Match but INCORRECT: 2/184  
No data found: 16/184

Gambar 4.3.2.1 Hasil stress testing menggunakan gambar *altered easy*

Match & CORRECT: 94/104  
Match but INCORRECT: 3/104  
No data found: 7/104

Gambar 4.3.2.2 Hasil stress testing menggunakan gambar *altered medium*

Match & CORRECT: 250/292  
Match but INCORRECT: 9/292  
No data found: 33/292

Gambar 4.3.2.3 Hasil stress testing menggunakan gambar *altered hard*

#### 4.4. Analisis Hasil Pengujian

Setelah melalui banyak uji coba dengan melakukan *stress testing* didapatkan bahwa untuk test case yang *altered easy* didapatkan bahwa persentase benar sebesar 90,2%, persentase salah sebesar 1,1%, dan persentase test case tidak ditemukan sebesar 8,7%, kemudian untuk *altered medium* didapatkan persentase benar sebesar 90,3%, persentase salah sebesar 3%, dan persentase tidak ditemukan sebesar 11%. Untuk *altered hard* didapatkan persentase benar sebesar 85%, persentase salah sebesar 3%, dan persentase tidak ditemukan sebesar 11%. Dari hasil ini, terlihat bahwa tingkat kesulitan gambar berpengaruh pada akurasi pencocokan sidik jari. Untuk kategori *altered easy*, persentase benar cukup tinggi dan persentase kesalahan rendah. Namun, seiring dengan meningkatnya tingkat kesulitan pada *altered medium* dan *altered hard*, terdapat peningkatan pada persentase kesalahan dan kasus yang tidak ditemukan. Penerapan algoritma KMP, BM, dan Levenshtein Distance dalam aplikasi pencocokan sidik jari GoyangSinggalang memungkinkan pencocokan yang akurat dan efisien. Hasil pengujian menunjukkan bahwa aplikasi ini mampu mencocokkan sidik jari dengan tingkat keberhasilan yang tinggi, terutama untuk gambar dengan tingkat kesulitan rendah hingga sedang.

# Bab 5

## Kesimpulan dan Saran

### 5.1 Kesimpulan

Algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) dapat digunakan sebagai pencarian sidik jari untuk keperluan autentifikasi. Algoritma Levenshtein distance dapat digunakan untuk mencari kemiripan antara dua string. Regular expression dapat digunakan untuk pencocokan string dengan efisien. Algoritma AES dapat dimanfaatkan untuk enkripsi penyimpanan data dalam database untuk melindungi data apabila bocor.

### 5.2 Saran

Untuk pengerjaan selanjutnya, kami menyarankan untuk mengeksplorasi algoritma lain untuk menghasilkan hasil yang lebih bagus, akurat dan terutama cepat untuk dieksekusi. Salah satu kekurangan program kami adalah waktu eksekusi (rata-rata ~3 detik). Dengan mengurangi waktu eksekusi pencocokan sidik jari, pengalaman pengguna akan meningkat drastis.

### 5.3 Tanggapan

Proyek tugas besar ini berjalan dengan lancar, memenuhi semua spesifikasi dari yang wajib hingga yang bersifat bonus. Kami mendapatkan pengalaman berharga dalam penerapan Pattern Matching yang telah kami pelajari selama kuliah, khususnya dalam konteks pencocokan sidik jari. Selain itu, kami juga mempelajari bahasa pemrograman baru, yaitu C#. Proyek ini memperkaya pemahaman kami tentang pentingnya keamanan data dan tantangan yang terkait dengan pengenalan sidik jari. Kami mengucapkan terima kasih kepada dosen kami, Pak Rinaldi Munir, atas bimbingannya selama perkuliahan, serta kepada para asisten IRK yang telah membantu kami dalam proses pengerjaan tugas ini.

### 5.4 Refleksi

Melalui proyek ini, kami belajar bahwa pengembangan sistem biometrik tidak hanya bergantung pada akurasi algoritma pencocokan, tetapi juga pada efisiensi dan keamanan penyimpanan data. Implementasi algoritma Knuth-Morris-Pratt dan Boyer-Moore dalam pencocokan sidik jari, serta penggunaan Levenshtein distance untuk pengukuran kemiripan, memberikan wawasan mendalam tentang kekuatan dan kelemahan masing-masing metode. Selain itu, penggunaan enkripsi AES untuk melindungi data dalam database menekankan pentingnya aspek keamanan

dalam sistem identifikasi. Tantangan terbesar yang kami hadapi adalah optimisasi waktu eksekusi, yang menyadarkan kami akan kebutuhan untuk terus mencari solusi yang lebih cepat dan efisien. Proyek ini juga mengajarkan kami pentingnya kolaborasi tim dan manajemen waktu dalam menyelesaikan tugas yang kompleks.

# Lampiran

Link *repository* GitHub :

[https://github.com/randyver/Tubes3\\_GoyangSinggalang](https://github.com/randyver/Tubes3_GoyangSinggalang)

Link video youtube :

[https://youtu.be/wyzfbE\\_4Sow?si=-9AkuN3wC7\\_sWsSx](https://youtu.be/wyzfbE_4Sow?si=-9AkuN3wC7_sWsSx)

## **Daftar Pustaka**

Laaksonen, A. (2018). Competitive Programmer's Handbook. Diakses pada 25 April 2024.

Munir, Rinaldi. 2024. “Pencocokan String (String/Pattern Matching)”.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>  
(Diakses pada 9 Juni 2024).

Khodra, Masayu Leylia. 2024. “String Matching dengan Regular Expression”.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf>  
(Diakses pada 9 Juni 2024).