

**LAPORAN TUGAS BESAR III  
IF2211 STRATEGI ALGORITMA**

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



Kelompok Selesaikan  
Disusun oleh:  
13522029              Ignatius Jhon Hezkiel Chan  
13522045              Elbert Chailes  
13522077              Enrique Yanuar

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG 2023**

## DAFTAR ISI

DAFTAR ISI.....	2
BAB 1	
DESKRIPSI TUGAS.....	4
BAB 2	
LANDASAN TEORI.....	8
2.1. Deskripsi Singkat Algoritma.....	8
2.1.1. Algoritma KMP (Knuth-Morris-Pratt).....	8
2.1.2. Algoritma BM (Boyer-Moore).....	9
2.1.3. Algoritma Regular Expression (Regex).....	10
2.2. Teknik Pengukuran Persentase Dengan Hamming Distance.....	11
2.3. Pengembangan Aplikasi Desktop.....	12
2.3.1. WPF (Windows Presentation Foundation).....	12
2.3.2. MySQL.....	13
2.3.3. Adminer (Database Management Tool).....	14
2.3.4. Docker.....	15
BAB 3	
ANALISIS PEMECAHAN MASALAH.....	17
3.1. Pemecahan Masalah.....	17
3.1.1. Preprocessing Gambar Sidik Jari.....	17
3.1.2. Strategi Penentuan Pattern dan Text untuk Pencocokan Sidik Jari.....	18
3.2. Fitur Fungsional Aplikasi.....	19
3.3. Contoh Ilustrasi Kasus.....	21
BAB 4	
IMPLEMENTASI DAN PENGUJIAN.....	23
4.1. Spesifikasi Teknis Program.....	23

4.2. Penjelasan Tata Cara Penggunaan Program.....	34
4.3. Hasil Pengujian.....	36
4.3.1. Pengujian menggunakan Algoritma KMP (Knuth-Morris-Pratt).....	37
4.3.2. Pengujian menggunakan Algoritma BM (Boyer Moore).....	39
4.3.3. Pencarian Sidik Jari dengan Pemilik Sidik Jari dengan Nama Terkorupsi.....	42
4.3.4. Pencarian dengan Basis Data Biodata yang Terenkripsi.....	42
4.4. Analisis hasil pengujian.....	43
<b>BAB 5</b>	
KESIMPULAN DAN SARAN.....	46
5.1. Kesimpulan.....	46
5.2. Saran.....	46
5.3. Tanggapan dan Refleksi.....	47
LAMPIRAN.....	49
Pranala Repository.....	49
Pranala Video.....	49
Dataset Fingerprint yang digunakan.....	49
DAFTAR PUSTAKA.....	50

# BAB 1

## DESKRIPSI TUGAS

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

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

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

Secara sekilas, penggunaan algoritma pattern matching dalam mencocokkan sidik jari terdiri atas tiga tahapan utama dengan skema sebagai berikut.



**Gambar 1.** Skema implementasi konversi citra sidik jari

Gambar yang digunakan pada proses pattern matching kedua algoritma tersebut adalah gambar sidik jari penuh berukuran  $m \times n$  pixel yang diambil sebesar 30 pixel setiap kali proses pencocokan data. Untuk tugas ini, Anda dibebaskan untuk mengambil jumlah pixel asalkan didasarkan pada alasan yang masuk akal (dijelaskan pada laporan) dan penanganan kasus ujung yang baik (misal jika ternyata ukuran citra sidik jari tidak habis dibagi dengan ukuran pixel yang dipilih). Selanjutnya, data pixel tersebut akan dikonversi menjadi *binary*. Karena binary hanya

memiliki variasi karakter satu atau nol, maka proses pattern matching akan membuat proses pencocokan karakter menjadi lambat karena harus sering mengulang proses pencocokan pattern. Cara yang dapat dilakukan untuk mengatasi hal tersebut adalah dengan mengelompokkan setiap baris kode biner per 8 bit sehingga membentuk karakter ASCII. Karakter ASCII 8-bit ini yang akan mewakili proses pencocokan dengan string data.

Untuk memberikan gambaran yang lebih jelas, berikut adalah contoh kasus citra sidik jari dan proses serta sampel hasil yang didapatkan. Dataset yang digunakan adalah dataset citra sidik jari yang terdapat pada dataset yang terdapat pada lampiran.

Citra Sidik Jari	Potongan nilai binary	Potongan ASCII 8-bits
	10100000101000001010000 01010000010100000101000 00101000001010000010100 00010100000101000001010 00001010000010100000101 00000101000001010000010 10000010100000101000001 01000001010000010100000 10100000101000001010000 0101000001010	iÿÿÿÿÿÿÿÿlÿÿü:Ãÿÿ<ÿÿuw ûÿkÿý,Pþÿz/ÿÿ'ÿ »ÿýD%?ÿ&%íÿÚá†È/Zíÿÿq %ÿúTý~Rûùt`íÿÿÿÿÿ iÿÿÿÿÿÿÿÿöÿÿy"bÿÿýÁüþÿÈ' b /ÿæ2ÿúYÿý\ ^ÿÿp fÿÿà !>ÿÿþÿÿÿÿÿÿ<ðÿÖ ÿÿQûÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ iÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ <>ÿÜþþ6Gÿÿ;Úÿÿfãÿÿ

**Gambar 2.** Hasil konversi citra sidik jari menjadi *binary* dan konversi ke ASCII 8-bits

Pada tahap implementasi di titik ini, telah dihasilkan serangkaian karakter ASCII 8-bit yang merepresentasikan sebuah sidik jari. Hasil ini yang akan dijadikan dasar pencarian sidik jari yang sama dengan daftar sidik jari yang terdapat pada basis data. Pencarian sidik jari yang paling mirip dengan sidik jari yang menjadi masukan pengguna dilakukan dengan algoritma pencocokan string *Knuth-Morris-Pratt* (KMP) dan *Boyer-Moore* (BM). Jika tidak ada satupun sidik jari pada basis data yang exact match dengan sidik jari yang menjadi masukan melalui algoritma KMP ataupun BM, maka gunakan sidik jari paling mirip dengan kesamaan diatas nilai tertentu. Anda diberikan kebebasan untuk menentukan nilai batas persentase kemiripan ini, silakan melakukan pengujian untuk menentukan nilai tuning yang tepat dan jelaskan pada laporan. Metode perhitungan tingkat kemiripan juga dibebaskan kepada Anda asalkan dijelaskan di laporan. Akan tetapi, asisten sangat menyarankan untuk menggunakan salah satu dari

algoritma *Hamming Distance*, *Levenshtein Distance*, ataupun *Longest Common Subsequence* (LCS).

Fungsi dari deteksi biometrik adalah mengidentifikasi seseorang, oleh sebab itu, pada proses implementasi program ini, sebuah citra sidik jari akan dicocokkan dengan biodata seseorang. Biodata yang dicocokkan terdiri atas data-data yang terdapat pada KTP, antara lain : NIK, nama, tempat/tanggal lahir, jenis kelamin, golongan darah, alamat, agama, status perkawinan, pekerjaan, dan kewarganegaraan. Relasi ini dibuat dalam sebuah basis data dengan skema detail seperti yang tertera pada bagian di bawah ini.

`biodata'	
PK	'NIK' varchar(16) NOT NULL
	'nama' varchar(100) DEFAULT NULL
	'tempat_lahir' varchar(50) DEFAULT NULL
	'tanggal_lahir' date DEFAULT NULL
	'jenis_kelamin' enum('Laki-Laki','Perempuan') DEFAULT NULL
	'golongan_darah' varchar(5) DEFAULT NULL
	'alamat' varchar(255) DEFAULT NULL
	'agama' varchar(50) DEFAULT NULL
	'status_perkawinan' enum('Belum Menikah','Menikah','Cerai') DEFAULT NULL
	'pekerjaan' varchar(100) DEFAULT NULL
	'kewarganegaraan' varchar(50) DEFAULT NULL

'sidik_jari'	
	'berkas_citra' text
	'nama' varchar(100) DEFAULT NULL

**Gambar 3.** Skema basis data yang digunakan

Seorang pribadi dapat memiliki lebih dari satu berkas citra sidik jari (relasi *one-to-many*). Akan tetapi, seperti yang dapat dilihat pada skema relasional di atas keduanya tidak terhubung dengan sebuah relasi. Hal ini disebabkan karena pada kasus dunia nyata, data yang disimpan bisa saja mengalami korup. Dengan membuat atribut kolom yang mungkin korup adalah atribut nama pada tabel biodata, maka atribut nama pada tabel sidik\_jari tidak dapat memiliki foreign-key yang mereferensi ke tabel biodata. Pada tugas besar kali ini, kita akan coba melakukan simulasi implementasi data korup yang hanya mungkin terjadi pada atribut nama di tabel biodata (asumsikan kolom lain pada setiap tabel tidak mengalami korup). Akan tetapi, karena tujuan utama program adalah mengenali identitas seseorang secara lengkap hanya dengan menggunakan sidik jari, maka harus dilakukan sebuah skema untuk menangani data korup tersebut.

Sebuah data yang korup dapat memiliki berbagai macam bentuk. Pada tugas ini, jenis data korup adalah bahasa alay Indonesia. Terdengar lucu, tetapi para pendahulu kita sudah membuat bahasa ini untuk berkomunikasi kepada sesamanya. Pada tugas besar ini, dibutuhkan

untuk menangani kombinasi dari tiga buah variasi bahasa alay, yaitu kombinasi huruf besar-kecil, penggunaan angka, dan penyingkatan.

Cara yang dapat digunakan untuk menangani ini adalah dengan menggunakan *Regular Expression* (Regex). Lakukan konversi pola karakter alay hingga dapat dikembalikan ke bentuk alfabetik yang bersesuaian. Setelah menggunakan Regex, Anda akan diminta kembali untuk melakukan pattern matching antara nama yang bersesuaian dengan algoritma KMP dan BM dengan ketentuan yang sama seperti saat pencocokan sidik jari.

Secara umum, berikut adalah cara umum penggunaan program yang hendak dikembangkan:

1. Pengguna terlebih dahulu memasukkan citra sidik jari yang ingin dicari biodatanya.
2. Setelah citra dimasukkan, pilih opsi pencarian, ingin melakukan pencarian apakah akan menggunakan algoritma KMP atau BM.
3. Tekan tombol search, program kemudian akan memproses, mencari citra sidik jari dari basis data yang memiliki kemiripan dengan citra sidik jari yang menjadi masukan.
4. Program akan menampilkan list biodata jika ditemukan citra sidik jari yang memiliki kemiripan dengan batas persentase tertentu. Program juga dapat mengeluarkan informasi tidak ada sidik jari yang mirip jika semua citra dalam basis data tidak memiliki kemiripan dengan masukan.
5. Terdapat informasi terkait waktu eksekusi program dan persentase kemiripan citra.



**Gambar 4.** Referensi tampilan UI *desktop-app*

## BAB 2

### LANDASAN TEORI

#### 2.1. Deskripsi Singkat Algoritma

##### 2.1.1. Algoritma KMP (*Knuth-Morris-Pratt*)

Algoritma *Knuth-Morris-Pratt* (KMP) adalah sebuah algoritma pencocokan string yang digunakan untuk mencari kecocokan pola (pattern) dalam sebuah teks. Algoritma ini ditemukan oleh Donald Knuth dan Vaughan Pratt, dengan perbaikan oleh Morris. KMP sangat efisien dalam mencari kecocokan pola dalam sebuah teks dengan kompleksitas waktu  $O(n + m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola.

Ide dasar di balik KMP adalah penggunaan informasi yang telah diperoleh dari kegagalan pencocokan sebelumnya untuk menghindari pencocokan kembali yang tidak perlu. Ini dilakukan dengan membangun tabel yang dikenal sebagai "tabel lompatan" atau "tabel kegagalan" yang mengindikasikan kemungkinan pergeseran pola yang mungkin terjadi saat pencocokan gagal.

Berikut adalah langkah-langkah umum dalam algoritma KMP:

1. Buat tabel lompatan yang akan membantu menentukan pergeseran pola saat pencocokan gagal.
2. Pencocokan dimulai dari awal teks dan pola. Saat mencocokkan, teks dan pola akan dibandingkan karakter per karakter.
3. Jika sebuah karakter tidak cocok, algoritma akan menggunakan tabel lompatan untuk menentukan pergeseran pola yang sesuai.
4. Proses berlanjut sampai seluruh pola cocok atau sampai akhir teks dicapai.

Dengan menggunakan tabel lompatan, KMP menghindari kebutuhan untuk memeriksa kembali setiap karakter dalam teks yang telah dicocokkan sebelumnya. Hal ini membuat algoritma KMP lebih efisien daripada algoritma pencocokan string yang sederhana, terutama ketika mencari pola dalam teks yang panjang.

Kelebihan algoritma KMP adalah bahwa algoritma ini tidak pernah perlu mundur ke belakang dalam teks input ( $T$ ). Hal ini membuat algoritma ini sangat baik untuk

memproses file yang sangat besar yang dibaca dari perangkat eksternal atau melalui aliran jaringan. Dengan tidak perlu mundur ke belakang, KMP dapat bekerja secara efisien dalam konteks di mana memori atau sumber daya lainnya mungkin terbatas.

Namun, algoritma KMP memiliki kelemahan ketika ukuran alfabet meningkat. Semakin besar alfabet, semakin besar peluang terjadinya ketidakcocokan (*mismatch*). Ketidakcocokan cenderung terjadi di awal pola, tetapi KMP menjadi lebih cepat ketika ketidakcocokan terjadi lebih lanjut.

Salah satu ekstensi dari algoritma KMP adalah bahwa versi dasarnya tidak memperhitungkan huruf dalam teks yang menyebabkan ketidakcocokan. Dengan kata lain, algoritma dasar tidak mempertimbangkan informasi tambahan tentang karakter-karakter yang sudah dicocokkan sebelumnya.

### **2.1.2. Algoritma BM (*Boyer-Moore*)**

Algoritma *pattern matching* Boyer-Moore adalah salah satu algoritma yang efisien dalam mencari kecocokan pola dalam sebuah teks. Algoritma ini berbasis pada dua teknik utama: teknik cermin (*looking-glass technique*) dan teknik loncatan karakter (*character-jump technique*).

#### **1. Teknik Cermin (*Looking-Glass Technique*)**

Algoritma Boyer-Moore memulai pencarian pola dengan memindai teks mundur melalui pola, dimulai dari akhir pola. Dengan cara ini, algoritma mencoba mencocokkan pola dengan teks mulai dari bagian akhir pola, bukan dari awalnya. Hal ini memungkinkan untuk menghindari beberapa perbandingan karakter yang tidak perlu.

#### **2. Teknik Loncatan Karakter (*Character-Jump Technique*)**

Ketika terjadi ketidakcocokan (*mismatch*) pada  $T[i] == x$ , di mana  $T$  adalah teks dan  $x$  adalah karakter dalam teks yang tidak cocok dengan karakter di posisi yang sesuai dalam pola  $P$ . Algoritma Boyer-Moore menggunakan teknik loncatan karakter untuk menentukan pergeseran pola yang sesuai. Ada tiga kemungkinan kasus yang dicoba secara berurutan:

- A. **Kasus 1.** Jika karakter dalam pola  $P$  yang tidak cocok dengan  $T[i]$  tidak ada dalam pola  $P$ , maka pola  $P$  dapat digeser sejauh panjang pola.

- B. **Kasus 2.** Jika karakter dalam pola P yang tidak cocok dengan  $T[i]$  ada dalam pola P, maka algoritma akan mencocokkan kembali karakter pola dengan karakter di teks yang sesuai dengan aturan pemindai karakter mundur.
- C. **Kasus 3.** Jika tidak ada karakter dalam pola P yang tidak cocok dengan  $T[i]$ , maka pola P diperkecil sampai karakter terakhir yang cocok dengan  $T[i]$ .

Algoritma Boyer-Moore biasanya memiliki kompleksitas waktu yang sangat baik, terutama dalam kasus-kasus di mana pola memiliki panjang yang relatif pendek dibandingkan dengan teks yang dicari. Namun, kompleksitas ruangnya bisa menjadi lebih tinggi dibandingkan dengan algoritma pencocokan string lainnya seperti Knuth-Morris-Pratt (KMP).

Kompleksitas ruang algoritma Boyer-Moore terutama bergantung pada tabel penyeimbang yang digunakan untuk menentukan pergeseran karakter dalam kasus ketidakcocokan (mismatch). Tabel ini biasanya membutuhkan ruang tambahan untuk disimpan, dan ukurannya tergantung pada ukuran alfabet yang digunakan dalam teks dan pola.

Secara umum, kompleksitas ruang algoritma Boyer-Moore adalah  $O(m + k)$ , di mana m adalah panjang pola dan k adalah ukuran alfabet. Ini karena algoritma ini membutuhkan tabel penyeimbang dengan ukuran  $m * k$  untuk menangani setiap kemungkinan karakter dalam alfabet pada setiap posisi dalam pola.

Meskipun kompleksitas ruangnya lebih tinggi dibandingkan dengan KMP, algoritma Boyer-Moore masih efisien dalam penggunaan memori dalam kebanyakan kasus praktis. Keunggulan utamanya terletak pada kompleksitas waktu yang rendah, yang membuatnya menjadi pilihan yang baik untuk pencocokan pola dalam teks besar atau dalam aplikasi di mana efisiensi waktu menjadi pertimbangan utama.

### 2.1.3. Algoritma Regular Expression (Regex)

Algoritma regex (regular expression) merupakan sebuah algoritma yang digunakan untuk mencocokkan pola teks dengan pola yang didefinisikan dalam bentuk regular expression. Regular expression adalah sekumpulan aturan yang digunakan untuk mencocokkan dan mengidentifikasi pola dalam teks.

Kompleksitas ruang dari algoritma regex bervariasi tergantung pada implementasinya dan juga pada pola regex yang digunakan. Beberapa implementasi algoritma regex menggunakan metode yang memerlukan ruang tambahan untuk menyimpan informasi tentang pola yang diberikan, sementara yang lain mungkin mencoba meminimalkan penggunaan memori semaksimal mungkin.

Umumnya, algoritma regex memerlukan ruang tambahan yang sebanding dengan panjang pola yang diberikan, karena perlu menyimpan informasi tentang pola tersebut untuk mencocokkannya dengan teks yang diberikan. Sebagai contoh, dalam implementasi algoritma regex yang menggunakan finite automata atau deterministic finite automata (DFA), tabel transisi atau state machine yang digunakan untuk mencocokkan pola akan memerlukan ruang tambahan yang sebanding dengan panjang pola.

Kompleksitas ruang juga bisa dipengaruhi oleh kompleksitas pola regex itu sendiri. Pola regex yang kompleks, seperti yang mengandung ekspresi reguler rekursif atau pola yang sangat panjang, cenderung memerlukan lebih banyak ruang dalam proses pencocokannya.

Secara umum, algoritma regex bisa memiliki kompleksitas waktu yang bervariasi antara  $O(n)$  hingga  $O(n^m)$ , di mana  $n$  adalah panjang teks dan  $m$  adalah panjang pola regex. Namun, kompleksitas tersebut bisa berubah tergantung pada faktor-faktor di atas. Penting untuk memahami karakteristik pola dan teks yang digunakan dalam aplikasi Anda saat mempertimbangkan kinerja algoritma regex.

## 2.2. Teknik Pengukuran Persentase Dengan *Hamming Distance*

*Hamming Distance* adalah sebuah metrik yang digunakan untuk mengukur perbedaan antara dua rangkaian (string) yang memiliki panjang yang sama. *Hamming Distance* mengukur berapa banyak posisi di mana karakter-karakter yang sesuai dalam kedua rangkaian berbeda satu sama lain.

Secara formal, *Hamming Distance* antara dua string dengan panjang yang sama,  $s$  dan  $t$ , adalah jumlah posisi di mana karakter-karakter yang sesuai pada posisi yang sama dalam kedua string berbeda.

$$\text{HammingDistance}(s, t) = \text{jumlah } \{i : s[i] \neq t[i]\}$$

Misalnya, jika kita memiliki dua string, yaitu “karpet” dan “kertas”. Kedua string tersebut memiliki panjang yang sama, yaitu 6. *Hamming Distance* antara kedua string tersebut adalah 3, karena terdapat tiga posisi di mana karakter pada posisi yang sama berbeda:

- Pada posisi 1: "k" (dari "karpet") berbeda dengan "k" (dari "kertas")
- Pada posisi 4: "p" (dari "karpet") berbeda dengan "t" (dari "kertas")
- Pada posisi 5: "e" (dari "karpet") berbeda dengan "a" (dari "kertas")

*Hamming Distance* berguna dalam berbagai konteks, termasuk dalam teori informasi, pengkodean, bioinformatika, dan komputasi. Misalnya, dalam bioinformatika, *Hamming Distance* dapat digunakan untuk membandingkan sekuen DNA atau protein. Semakin rendah *Hamming Distance* antara dua sekuen, semakin mirip kedua sekuen tersebut. Namun, pada tugas besar kali ini, *hamming distance* digunakan untuk mengukur kemiripan sidik jari yang paling *matching*, maka perlu dicari *hamming distance* yang paling kecil.

### **2.3. Pengembangan Aplikasi Desktop**

Aplikasi Desktop yang dikembangkan pada tugas besar ini menggunakan bahasa pemrograman C# dengan menggunakan *framework WPF (Windows Presentation Foundation)*. Sedangkan, untuk algoritma-algoritma untuk menyusun aplikasi *desktop* ini ditranslasi menjadi dalam bentuk kode dan disimpan dalam kelasnya masing-masing dengan bahasa pemrograman C#. Untuk menyimpan informasi sidik jari dan biodata dalam jumlah masif, aplikasi *desktop* menggunakan basis data *MySQL* dengan menggunakan dengan menggunakan *Adminer* sebagai *database management system* dalam proses pengembangan aplikasi *desktop* ini. Untuk membantu migrasi aplikasi ini dari satu pengguna ke pengguna lain, juga terdapat *docker* dalam pengembangan aplikasi ini. Berikut merupakan rincian terkait teknologi-teknologi yang dipakai dalam pengembangan aplikasi ini.

#### **2.3.1. WPF (*Windows Presentation Foundation*)**

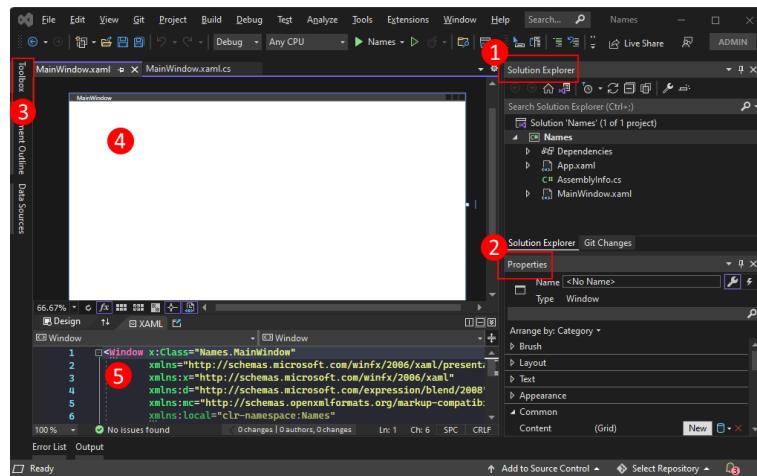
*Windows Presentation Foundation* (WPF) adalah sebuah teknologi dari *Microsoft* yang digunakan untuk membangun aplikasi *desktop Windows* dengan antarmuka pengguna yang kaya dan menarik secara visual. WPF memberikan pendekatan

deklaratif dalam membangun antarmuka pengguna (UI), yang memungkinkan pengembang untuk mendefinisikan tata letak, gaya, animasi, dan interaksi UI menggunakan bahasa markup yang dikenal sebagai XAML (*eXtensible Application Markup Language*).

XAML, atau *eXtensible Application Markup Language*, adalah bahasa yang memungkinkan kita mendefinisikan tampilan antarmuka pengguna (UI) dalam aplikasi desktop *Windows* dengan cara yang deklaratif. Ini berarti, daripada menentukan tampilan UI secara imperatif dalam kode program, kita dapat mendeskripsikannya dalam file teks yang terstruktur.

Satu hal yang membuat XAML sangat kuat adalah kemampuannya untuk *data binding*. Ini berarti dapat menghubungkan data dari sumber eksternal, seperti database atau objek .NET, ke elemen-elemen UI.

Dalam pengembangan aplikasi yang akan dijelaskan pada laporan ini, penggunaan WPF untuk membantu dan memudahkan proses pengembangan GUI aplikasi dengan salah satu fitur yang disediakan, yaitu *drag and drop feature* dan ditambah dengan penambahan *styling* yang dilakukan langsung pada kode XAML yang memberikan fleksibilitas yang lebih.



**Gambar 2.3.1.1.** Tampilan penggunaan WPF pada *Visual Studio*

### 2.3.2. MySQL

MySQL adalah salah satu sistem manajemen basis data (DBMS) relasional yang paling populer dan sering digunakan di dunia. Berikut adalah beberapa poin penting terkait dengan MySQL:

1. *Open-Source*. MySQL dikembangkan sebagai proyek sumber terbuka, yang berarti kode sumbernya tersedia untuk umum dan dapat diunduh dan digunakan secara gratis. Ini membuat MySQL sangat populer di kalangan pengembang perangkat lunak open-source dan komunitas pengembang.
2. Relasional. MySQL adalah DBMS relasional, yang berarti data disimpan dalam bentuk tabel yang terdiri dari baris dan kolom. Ini memungkinkan untuk melakukan kueri kompleks dan menyimpan data secara terstruktur.
3. *Multi-Platform*. MySQL tersedia untuk berbagai platform, termasuk Windows, Linux, macOS, dan berbagai sistem operasi lainnya. Ini memungkinkan pengembang untuk menggunakan MySQL di berbagai lingkungan pengembangan.
4. Bahasa SQL. MySQL menggunakan SQL (Structured Query Language) sebagai bahasa untuk melakukan operasi terhadap basis data, seperti pengambilan data, penyisipan, pembaruan, dan penghapusan. MySQL mendukung sebagian besar fitur standar SQL dan juga memiliki ekstensi dan fitur tambahan yang khusus untuk MySQL.
5. Skalabilitas. MySQL dirancang untuk menangani beban kerja yang besar dan skalabilitas horizontal serta vertikal. Ini berarti MySQL dapat dengan mudah diperluas dengan menambahkan lebih banyak sumber daya ke server tunggal atau dengan mendistribusikan beban kerja ke beberapa server.
6. Kinerja Tinggi. MySQL dikenal karena kinerjanya yang tinggi dan kemampuan untuk menangani beban kerja tinggi dengan cepat. Ini dicapai melalui optimasi internal, pengindeksan data, dan fitur-fitur caching yang efisien.

Secara keseluruhan, MySQL adalah salah satu sistem manajemen basis data relasional yang paling populer dan kuat di dunia, digunakan oleh berbagai organisasi dan aplikasi untuk menyimpan, mengelola, dan mengakses data dengan efisien.

#### **2.3.3. Adminer (*Database Management Tool*)**

*Adminer* adalah sebuah alat manajemen basis data yang ringan dan mudah digunakan, sering digunakan oleh *developer* dan *administrator* basis data untuk mengelola sistem manajemen basis data (DBMS). Tampilan *Adminer* dapat dilihat seperti berikut.

**Gambar 2.3.3.1.** Tampilan Adminer

Dalam pengembangan aplikasi ini, *Tool Adminer* digunakan untuk membantu proses penyusunan basis data. Dengan *tool* ini, pengembang dapat melihat basis data yang terdapat pada MySQL secara lebih baik dan lebih *friendly-look* sehingga dapat memahami basis data dan kontennya secara lebih jelas. Penambahan, penghapusan, ataupun interaksi lainnya terhadap basis data juga menjadi lebih mudah dengan bantuan kertas *Adminer* ini.

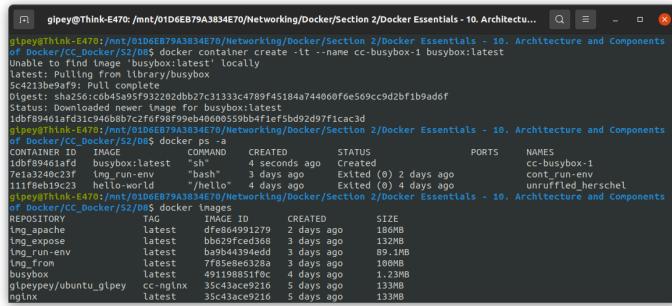
#### 2.3.4. Docker

*Docker* adalah *open-source platform* yang digunakan untuk membangun, mengirim, dan menjalankan aplikasi di lingkungan yang terisolasi yang disebut sebagai *container*. *Container* adalah lingkungan yang independen dan terisolasi yang membungkus perangkat lunak, termasuk kode, *runtime*, alat sistem, dan *library*, yang diperlukan untuk menjalankan aplikasi ke dalam satu paket.

*Container* dapat digunakan di berbagai lingkungan yang mendukung *Docker*, termasuk lingkungan pengembangan lokal, server fisik, *virtual machine*, dan *cloud*. Ini membuat aplikasi dapat dengan mudah dipindahkan dari lingkungan pengembangan ke lingkungan produksi tanpa perlu mengubah kode atau konfigurasi.

Pada proses pengembangan aplikasi ini, penggunaan *docker* dilakukan dengan tujuan untuk memudahkan migrasi aplikasi pada sebuah komputer menuju komputer lain

yang ingin menjalankan program. Ini disebabkan *docker* yang telah membantu membungkus aplikasi menjadi satu dari tampilan aplikasi (GUI), logika aplikasi, dan koneksi terhadap basis data. Dengan itu, pengguna hanya perlu melakukan *docker compose up* untuk menjalankan *image* pada *docker* yang terinstal pada komputer pengguna. Dengan ini, maka kompilasi atau kegiatan untuk menjalankan aplikasi yang kompleks akan menjadi lebih mudah dan simpel karena hanya perlu menjalankan suatu perintah dan *docker* akan meng-*handle* sisa pekerjaannya.



```

gipey@Think-E470: /mnt/01D6EB79A3834E70/Networking/Docker/Section 2/Docker Essentials - 10. Architecture and Components of Docker/CC_Docker/52/0BS$ docker container create -it --name cc-busybox-1 busybox:latest
Unable to find image 'busybox:latest' locally
Latest: Pulling from library/busybox
5c83a2f19f93: Pulling [REDACTED]
Digest: sha256:cdb45a95f932202bb27c31333c4789f45184a74400bf0e569cc9d2bfb1b9adff
Status: Downloaded newer image for busybox:latest
1dbf89461af1df31c46bbb7c2f6f9f9f9e4d000559b4f41ef5bd92d97fcac3d
gipey@Think-E470: /mnt/01D6EB79A3834E70/Networking/Docker/Section 2/Docker Essentials - 10. Architecture and Components of Docker/CC_Docker/52/0BS$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
1dbf89461af1d busybox:latest "sh" 4 seconds ago Created cc-busybox-1
7e1a3240c23f lmg_run-env "bash" 3 days ago Exited (0) 2 days ago cont_run-env
111f8eb19c2d lmg_run-env "bash" 4 days ago Exited (0) 4 days ago untried_herschel
gipey@Think-E470: /mnt/01D6EB79A3834E70/Networking/Docker/Section 2/Docker Essentials - 10. Architecture and Components of Docker/CC_Docker/52/0BS$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
lmg_apache latest dfe84441270 2 days ago 180MB
lmg_expose latest b030444ec0368 3 days ago 130MB
lmg_run-env latest ba9644394eddd 3 days ago 89.1MB
lmg_fron latest 7f85e66328a 3 days ago 100MB
busybox latest 491198851fb 4 days ago 1.23MB
gipey/ubuntu_gipey cc-nginx 35c43ace9210 5 days ago 133MB
nginx latest 35c43ace9210 5 days ago 133MB

```

**Gambar 2.3.4.1.** Tampilan penggunaan *docker* pada terminal

## BAB 3

### ANALISIS PEMECAHAN MASALAH

#### 3.1. Pemecahan Masalah

##### 3.1.1. Preprocessing Gambar Sidik Jari

Gambar *fingerprint* akan dilakukan preprocess agar dapat dilakukan operasi string matching penentuan sidik jari yang cocok. Hasil dari preprocess gambar sidik jari masukan pengguna sebagai pattern dari matching dan hasil preprocess gambar data pada database sebagai text dari matching. Tahap-tahapan preprocess gambar fingerprint adalah sebagai berikut.

1. Gambar akan diubah menjadi suatu bitmap warna, untuk mendapatkan nilai RGB dari tiap pixel pada gambar.
2. Bitmap warna tersebut diubah menjadi bitmap yang menyimpan nilai grayscale dari tiap pixel. Nilai grayscale ini didapatkan dengan rumus  $\text{grayscale} = (\text{RedScale} * 0.3) + (\text{GreenScale} * 0.59) + (\text{BlueScale} * 0.11)$ .
3. Kemudian bitmap grayscale tersebut diubah menjadi bitmap binary, dimana nilai 255 menandakan hitam dan nilai 0 menandakan putih. Penentuan nilai biner suatu pixel dilihat dari nilai grayscale, apakah berada di range 0-127 atau 128-255. Jika grayscale berada di range 0-127, maka nilai binary adalah 0. Sebaliknya jika berada di range 128-255, maka nilai binary adalah 255.
4. Bitmap grayscale tersebut diubah menjadi string biner, dengan melakukan append nilai biner tiap pixel pada string hasil secara berurutan dari kiri atas hingga kanan bawah. Nilai biner dari suatu pixel adalah 1 jika nilai bitmap binernya 255, dan 0 jika nilai bitmapnya 0.
5. Mengubah string biner yang telah didapatkan menjadi string ASCII character.

ASCII character ini didapatkan dengan mengambil nilai tiap 8 bit biner dan mengubahnya menjadi karakter yang sesuai. Untuk preprocess gambar input pengguna sebagai pattern, akan diambil dua blok 32 bit biner dengan transisi nilai terbanyak (akan dijelaskan pada bagian berikutnya) untuk diubah menjadi ASCII, dan untuk data pada database akan diubah seluruh bagian string biner yang didapatkan sebelumnya, menjadi ASCII.

Hasil *preprocess* gambar menjadi string ASCII ini yang akan digunakan sebagai pattern dan text pada string matching pencocokan sidik jari. Pattern akan berupa string karakter ASCII dari gambar input pengguna sedangkan text berupa string karakter ASCII gambar sidik jari pada database. Penggunaan string ASCII dibandingkan string biner didasarkan pada alasan optimasi karena dengan ASCII, panjang string pencocokan dikurang menjadi  $\frac{1}{8}$  kali panjang string biner awal sehingga menjadi lebih singkat dan optimal.

### 3.1.2. Strategi Penentuan Pattern dan Text untuk Pencocokan Sidik Jari

Penentuan pattern dan text yang digunakan pada operasi pencocokan sidik jari merupakan kunci dari keberhasilan pencocokan. Pattern dan text yang akan dilakukan operasi matching diharapkan dapat mewakili karakteristik sidik jari yang terkait sehingga dapat memaksimalkan hasil pencarian sidik jari. Suatu sidik jari dari database dikatakan cocok dengan sidik jari input pengguna jika pattern ASCII dari sidik jari input ditemukan pada text ASCII dari sidik jari pada database (exact match), ataupun jika memiliki *Hamming Distance* terkecil kalau tidak ditemukan exact match.

Pattern yang akan digunakan pada pencocokan ditentukan dari “kualitas” suatu blok string biner berukuran 32 bit. Kualitas disini adalah jumlah transisi biner dari 0 ke 1 ataupun sebaliknya. String biner dengan jumlah transisi nilai bit terbesar menandakan daerah pada sidik jari yang padat akan garis dan jarak, yang diharapkan menempati bagian terpadat akan garis yang menjadi keunikan suatu sidik jari.

Pemilihan ukuran 32 bit juga dikarenakan memerlukan jumlah bit yang kelipatan 8 agar dapat diubah menjadi ASCII. Ukuran ini juga dipilih agar jumlah pixel pada gambar habis dibagi dengan ukuran blok. Referensi ukuran dilihat dari data gambar pada database (96px\*103px) dan untuk menyamakan data gambar yang dimiliki dengan input pengguna, gambar sidik jari input pengguna juga di resize sesuai ukuran tersebut. Untuk data gambar pada database yang tidak berukuran dengan dimensi tersebut, juga akan diubah menjadi ukuran tersebut sebelum dilakukan *preprocess*.

Blok string biner yang akan menjadi kandidat pattern diambil tiap jarak 8 bit sebanyak 32 bit (contoh ilustrasi pada bagian 3.3). Dengan berdasar pada nilai kualitas

dari blok string biner, kita mengambil 2 blok biner berukuran 32 bit terbaik sebagai pattern dan mengubahnya menjadi string ASCII character sehingga kita akan memiliki 2 kandidat pattern yang masing-masing berukuran 4 karakter ASCII.

### 3.1.3. Proses Pencarian Solusi dengan Algoritma KMP dan Algoritma BM

Setelah mendapatkan 2 pattern tadi, kita lakukan operasi string matching kedua pattern tersebut terhadap text yang berupa string ASCII character, pada tiap data sidik jari pada database. Algoritma operasi string matching ini dapat berupa algoritma KMP ataupun algoritma BM. Untuk implementasi KMP dan BM tidak ada perubahan dengan algoritma umum. Iterasi akan dilakukan dari data awal hingga data akhir. Jika salah satu pattern ditemukan pada string ASCII data, maka sidik jari yang cocok telah ditemukan. Data sidik jari tersebut kemudian dimunculkan pada layar

Jika operasi pencocokan string terhadap tiap data tidak ada yang menghasilkan exact match, kita cari nilai *Hamming Distance* antara gambar input pengguna dengan data sidik jari pada database. Pada bagian ini, untuk gambar input pengguna dan gambar data pada database sama-sama menggunakan string ASCII keseluruhan gambar. Nilai *Hamming Distance* ini dilakukan antara tiap pixel gambar yang memiliki posisi sama. Setelah didapatkan nilai HD dari tiap data, ambil data sidik jari yang memiliki nilai HD terkecil. Data dari sidik jari tersebut kemudian dimunculkan pada layar.

## 3.2. Fitur Fungsional Aplikasi

Dalam pembangunan, tentu saja terdapat beberapa fitur yang harus dipenuhi sesuai dengan rancangan dan tujuan awal yang hendak dicapai dalam pembangunan aplikasi ini. Aplikasi ini dirancang dengan tujuan untuk mencari sidik jari yang paling cocok dengan sidik jari yang dimasukkan oleh pengguna, beserta dengan biodata pemilik sidik jari tersebut. Berikut ini adalah rincian fitur-fitur yang disediakan dalam implementasi aplikasi.

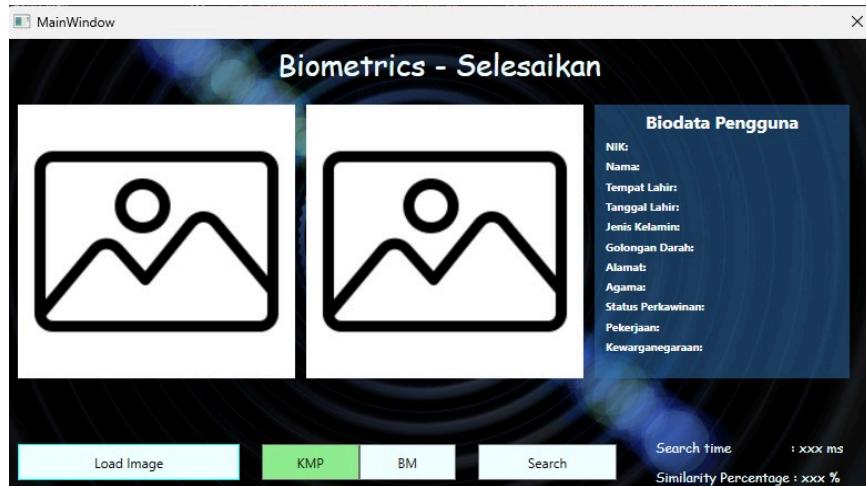
- Aplikasi ini memungkinkan penerimaan gambar sebagai input dari pengguna, mendukung format file .BMP, dan menampilkan gambar tersebut sebagai bagian dari hasil inputan. Pengguna juga dapat melakukan input dengan mekanisme *drag and drop*.
- Pengguna dapat memilih antara dua algoritma, yaitu BM (Boyer-Moore) atau KMP (Knuth-Morris-Pratt), melalui opsi *toggle button* yang disediakan.

- Fitur pencarian di dalam aplikasi ini didukung oleh sebuah database yang berfungsi sebagai basis pencarian. Fungsi pencarian dipanggil untuk memproses gambar inputan menggunakan bahasa pemrograman C#.
- Hasil pencarian ditampilkan secara lengkap, termasuk informasi detail dari hasil pencarian dan gambar yang memiliki kemiripan tinggi dengan gambar yang di input oleh pengguna. Terdapat juga informasi berupa lama pencarian sidik jari dalam format (milisekon atau sekon), serta tingkat kemiripan antar sidik jari yang didapatkan setelah dilakukan pencarian.
- Untuk mempercepat pencarian sidik jari yang dilakukan terhadap basis data yang mengandung banyak data yang tidak sedikit jumlahnya, aplikasi memiliki sistem *caching* yang dapat mempercepat pencarian jauh dari sebelumnya ketika pencarian seluruh basis data telah dilakukan sebelumnya. Pencarian pertama mungkin akan mempunyai waktu pencarian yang normal, namun pencarian selanjutnya akan memiliki waktu pencarian yang jauh lebih cepat karena sistem *caching* yang diimplementasi.
- Aplikasi ini dapat membaca nama-nama dalam biodata yang terkorupsi datanya, atau dengan kata lain nama-nama orang yang tersimpan dalam bahasa alay. Telah tersedia fitur untuk membaca nama alay tersebut sehingga dapat dilakukan pendekatan terhadap nama yang sebenarnya.
- Selain itu, aplikasi juga menyediakan fitur enkripsi terhadap data-data setiap orang yang terdaftar dalam basis data Biodata dengan menggunakan algoritma enkripsi *blowfish*. Dengan adanya fitur ini, maka dapat dipastikan sekuritas data-data orang yang terdaftar dalam basis data dapat dijamin. Pengakses atau dekripsi data tersebut hanya dapat dilakukan dengan *key* yang telah di-*generate* ketika proses enkripsi.

Aplikasi ini juga memiliki fitur GUI (*Graphical user Interface*) yang dibuat dengan menggunakan bantuan kakas WPF. Dalam GUI ini, terdapat tombol-tombol yang interaktif sehingga memudahkan pengguna untuk memahami cara penggunaannya untuk pertama kalinya. Terdapat tombol-tombol seperti berikut:

- Tombol “Load Image” untuk menambahkan gambar sidik jari yang hendak dicari dalam basis data.
- Tombol *toggle KMP/BM* untuk memilih pencarian dengan menggunakan jenis algoritma yang mana, algoritma KMP (Knuth-Morris-Pratt) atau algoritma BM (Boyer Moore).

- Tombol “Search” untuk mulai melakukan pencocokan sidik jari terhadap basis data.



**Gambar 3.2.1.** Tampilan GUI Aplikasi

### 3.3. Contoh Ilustrasi Kasus

Misalkan input pengguna berupa gambar sidik jari dikonversi menjadi biner seperti sebagai berikut.

```
101000001010000010100000101000001010
000010100000101000001010000010100000
101000001010000010010100
```

#### 1. Penentuan Nilai kualitas dari blok string biner

String biner tersebut berukuran  $32 \times 3$  bit. Didapatkan blok-blok bit kandidat sebagai beserta nilai transisi nya sebagai berikut.

```
10100000101000001010000010100000 (15 transisi)
```

10100000101000001010000010010100 (17 transisi)

## 2. Mengambil Dua Pattern Terbaik

Memilih dua pattern 32 bit dengan jumlah transisi terbanyak, yakni

10100000101000001010000010010100 (17 transisi)

10100000101000001010000010100000 (15 transisi).

## 3. Mengubah Pattern menjadi ASCII

- 10100000101000001010000010010100 menjadi ♦♦♦?
- 10100000101000001010000010100000 menjadi ♦♦♦♦

Ctt : Nilai yang melebihi batas ASCII (127) akan diubah sesuai unicode koresponden dengan nilai tersebut

## 4. Mencocokkan Pattern dengan Text Data

Misalkan terdapat data sidik jari pada database dengan string biner sebagai berikut.

10100000101000001010000 01010000010100000101000 00101000001010000010100  
00010100000101000001010 00001010000010100000101 00000101000001010000010  
10000010100000101000001 01000001010000010100000 10100000101000001010000  
0101000001010

String biner ini kemudian diubah menjadi string ASCII sebagai berikut.

iÿÿÿÿÿÿÿÿÿÿù:Ãÿÿ<ÿÿuw      ûÿkÿý,Pþÿÿ/ÿÿ'      »ÿÿĐ³/ÿ&/4j¤Úa+È/Zíÿÿq  
½ÿúTý~Rûût'íÿÿÿÿ    iÿÿÿÿÿÿÿöÿÿ"bÿýyÁüþýÊ'    þ /ÿœ2ÿúYÿÿ\ ^ÿÿp    fÿÿà  
!ÿÿþþÿÿÿÿÿþ    <ðÿÖ    ²ÿÿQûÿ N°ÿášÿÿÿÿÿ    iÿÿÿÿÿÿÿÿÿðrÛðÿÿC¹/ÿÿ‡ÿ  
<ÿÛþþ6Gÿÿ;Úÿÿÿfàÿÿ

Kita lakukan string matching dengan kedua pattern pada text ASCII sidik jari pada database. Jika didapatkan pencocokan pattern dengan bagian dari text ASCII, maka hasil sidik jari yang cocok telah didapatkan. Jika tidak ada data sidik jari pada database yang cocok, maka diambil data sidik jari database yang memiliki nilai *Hamming Distance* terkecil dengan text string ASCII gambar sidik jari input pengguna.

## **BAB 4**

### **IMPLEMENTASI DAN PENGUJIAN**

#### **4.1. Spesifikasi Teknis Program**

Dalam membangun aplikasi ini, tentu saja terdapat beberapa komponen yang harus dibuat secara terpisah dan pada akhirnya digabung menjadi satu untuk menjadi satu aplikasi yang memenuhi fungsionalitas yang telah ditentukan. Dalam pembangunan aplikasi pencocokan sidik jari ini, digunakan beberapa jenis *tools* dimulai dengan bahasa pemrograman C# sebagai *base*, WPF dengan XAML untuk membangun GUI aplikasi, pembangunan koneksi dengan basis data MySQL, serta penyusunan atau *setup* Docker untuk membantu memudahkan *setup database* pada komputer yang berbeda-beda, namun tetap dengan mekanisme yang sama.

Dengan begitu, maka struktur *file* dalam *repository* kode untuk membangun kode ini menjadi kompleks dan modular sehingga pekerjaan secara paralel dapat dilakukan. Berikut merupakan penjelasan beberapa *file* utama dalam *repository* ini.

a. Folder *Algorithm*

i. File Bm.cs

File ini terdapat pada folder *algorithm* karena file ini berisi algoritma *boyer moore* untuk melakukan pencocokan string. Algoritma dienkapsulasi dalam sebuah kelas bernama Bm. Di dalam kelas Bm ini, terdapat sebuah metode statik sehingga metode ini dapat diakses oleh program utama di luar kelas Bm ini tanpa melakukan instansiasi kelas ini. Algoritma dapat dilihat pada **Gambar 4.1.1**.

```
namespace Selesaikan.Algorithm;

1 reference
public class Bm
{
    1 reference
    public static int Search(string text, string pat) {
        int n = text.Length;
        int m = pat.Length;
        int skip;
        var right = new Dictionary<char, int>();

        // Build the skip table
        for (int i = 0; i < m; i++) {
            right[pat[i]] = i;
        }

        for (int i = 0; i <= n - m; i += skip) {
            skip = 0;
            for (int j = m - 1; j >= 0; j--) {
                if (pat[j] != text[i + j]) {
                    int val = -1;
                    if(right.ContainsKey(text[i+j])){
                        val = right[text[i+j]];
                    }
                    skip = Math.Max(1, j - val);
                    break;
                }
            }
            if (skip == 0) {
                Console.WriteLine("Pattern found at position " + i);
                return i; // Found
            }
        }

        return -1; // Not found
    }
}
```

**Gambar 4.1.1.** Cuplikan kode kelas algoritma BM (*Boyer Moore*)

ii. File Hd.cs

File ini berisi kelas Hd (*Hamming Distance*) yang berisi sebuah metode statik untuk menghitung *hamming distance* / banyak karakter yang berbeda di antara dua *string* yang dibandingkan. Cuplikan kode dapat dilihat pada **Gambar 4.1.2.**

```
1  namespace Selesaikan.Algorithm;
2
3  public class Hd
4  {
5      public static int Calculate(string s1, string s2) {
6          if (s1.Length != s2.Length) {
7              throw new ArgumentException("Strings must be of equal length");
8          }
9          int distance = 0;
10         for (int i = 0; i < s1.Length; i++) {
11             if (s1[i] != s2[i]) {
12                 distance++;
13             }
14         }
15         return distance;
16     }
17 }
```

**Gambar 4.1.2.** Cuplikan kode kelas algoritma HD (*Hamming Distance*)

### iii. File Kmp.cs

File ini berisi kelas bernama Kmp yang berisi metode statik berupa KmpSearch dan ComputeLpsArray. Tujuan dibuat statik kedua metode ini adalah agar metode yang terdapat dalam kelas ini dapat digunakan oleh kelas yang berada di luar kelas ini tanpa harus melakukan instansiasi terlebih dahulu. Metode ComputeLpsArray merupakan sebuah metode untuk menghitung *Longest Proper Prefix* yang terdapat pada sebuah pattern, yang berfungsi sebagai metode *helper* untuk membantu perhitungan KmpSearch. Kemudian, metode KmpSearch merupakan sebuah metode yang berisi logika pencocokan string dengan menggunakan algoritma KMP. Tujuan dari algoritma KMP ini sama seperti BM, yaitu untuk menemukan *pattern* yang match di dalam sebuah *subset of string* dari sebuah *long text*. Hasil pengembalian dari metode KmpSearch berupa index ditemukannya *pattern* pada *long text*. Cuplikan kode algoritma ComputeLpsArray dan KmpSearch dapat dilihat pada **Gambar 4.3.1**.

```

using System.Security.Cryptography;
namespace Selesaikan.Algorithm;

1 reference
public class Kmp
{
    1 reference
    public static int KmpSearch(string text, string pattern) {
        int n = text.Length;
        int m = pattern.Length;
        int[] lps = new int[m];
        int j = 0; // Length of previous longest prefix suffix

        // Preprocess the pattern to calculate lps array
        ComputeLpsArray(pattern, m, lps);

        int i = 0;
        while (i < n) {
            if (pattern[j] == text[i]) {
                j++;
                i++;
            }
            if (j == m) {
                Console.WriteLine("Found pattern at index " + (i - j));
                // j = lps[j - 1];
                return i;
            }
            else if (i < n && pattern[j] != text[i]) {
                if (j != 0)
                    j = lps[j - 1];
                else
                    i = i + 1;
            }
        }
        return -1;
    }

    1 reference
    private static void ComputeLpsArray(string pattern, int m, int[] lps) {
        int length = 0;
        int i = 1;
        lps[0] = 0; // lps[0] is always 0

        // Loop calculates lps[i] for i=1 to m-1
        while (i < m) {
            if (pattern[i] == pattern[length]) {
                length++;
                lps[i] = length;
                i++;
            } else {
                if (length != 0) {
                    length = lps[length - 1];
                } else {
                    lps[i] = 0;
                    i++;
                }
            }
        }
    }
}

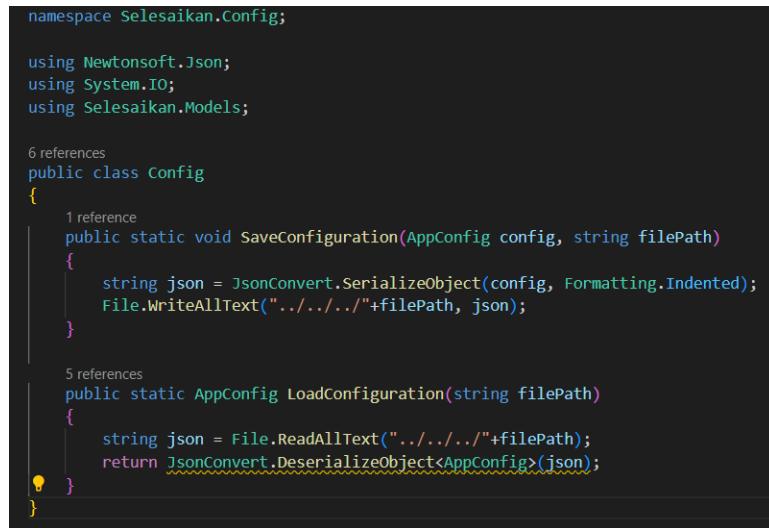
```

**Gambar 4.1.3.** Cuplikan kode kelas algoritma KMP (*Knuth-Morris-Pratt*)

#### b. Folder *Config*

Folder ini berisi file *config.cs*. Kelas Config yang bertanggung jawab untuk operasi penyimpanan dan pemenuhan konfigurasi aplikasi. Fungsi-fungsi dalam kelas ini

memanfaatkan *library* `Newtonsoft.Json` untuk serialisasi dan deserialisasi objek konfigurasi, yang memungkinkan penyimpanan dan pemulihan state konfigurasi aplikasi dalam format JSON. Metode `saveConfiguration` bertujuan untuk menyimpan objek konfigurasi ke dalam sebuah file, sedangkan metode `LoadConfiguration` bertujuan untuk untuk memuat konfigurasi aplikasi dari sebuah file. Cuplikan kode kelas `config` dapat dilihat pada **Gambar 4.1.4.**



```
namespace Selesaikan.Config;

using Newtonsoft.Json;
using System.IO;
using Selesaikan.Models;

6 references
public class Config
{
    1 reference
    public static void SaveConfiguration(AppConfig config, string filePath)
    {
        string json = JsonConvert.SerializeObject(config, Formatting.Indented);
        File.WriteAllText("../../" + filePath, json);
    }

    5 references
    public static AppConfig LoadConfiguration(string filePath)
    {
        string json = File.ReadAllText("../../" + filePath);
        return JsonConvert.DeserializeObject<AppConfig>(json);
    }
}
```

**Gambar 4.1.4.** Cuplikan kode kelas Config pada `config.cs`

#### c. Folder *Database*

Folder ini berisi file `Database.cs` yang berisi segala fungsi yang digunakan oleh aplikasi untuk berinteraksi dengan *database*, serta fungsi untuk melakukan koneksi dengan basis data *MySql* yang sudah di-setup pada awal inisialisasi proyek. Di dalam kelas Database ini berisi metode konstruktor untuk inisialisasi kelas Database dan membaca string koneksi dari file konfigurasi dan menginisialisasi koneksi database, fungsi `getSidikJari` untuk mengambil semua entri sidik jari dari database, fungsi `getBiodata` untuk mengambil biodata berdasarkan nama dari database, fungsi `getAllBiodata` untuk mengambil semua biodata dari database, fungsi `TruncateBiodata` untuk mengosongkan (*truncate*) tabel biodata di database, dan metode `SaveAllBiodata` untuk menyimpan sejumlah biodata ke dalam database. Cuplikan kode dapat dilihat pada **Gambar 4.1.5.**

```

using System.Text;
using MySql.Data.MySqlClient;
using Selesaikan.Models;

namespace Selesaikan.Database
{
    3 references
    public class Database
    {
        6 references
        private readonly string _connectionString;

        1 reference
        public Database()
        {
            Console.WriteLine("aaa");

            AppConfig loadedConfig = Config.Config.LoadConfiguration("config.json");
            Console.WriteLine([loadedConfig.DatabaseConnectionString]);
            _connectionString = loadedConfig.DatabaseConnectionString;
            Console.WriteLine("bbb");
        }

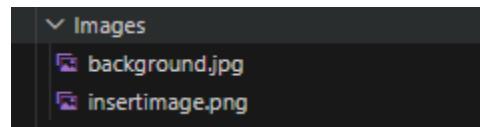
        1 reference
        public List<SidikJari> GetSidikJari()
        {
            List<SidikJari> dataSidikJari = new List<SidikJari>();
            try
            {
                using (MySqlConnection connection = new MySqlConnection(_connectionString))
                {
                    connection.Open();
                    string sql = "SELECT nama, berkas_citra FROM sidik_jari";
                    MySqlCommand command = new MySqlCommand(sql, connection);
                    using (MySqlDataReader reader = command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            dataSidikJari.Add(new SidikJari()
                            {
                                Nama = reader.GetString(0),
                                BerkasCitra = reader.GetString(1),
                            });
                        }
                    }
                }
            catch (MySqlException e)
            {
                // Handle exceptions related to SQL here
                Console.WriteLine("A SQL error occurred: " + e.Message);
            }
            catch (Exception e)
            {

```

**Gambar 4.1.5.** Cuplikan kode kelas Database

#### d. Folder *Images*

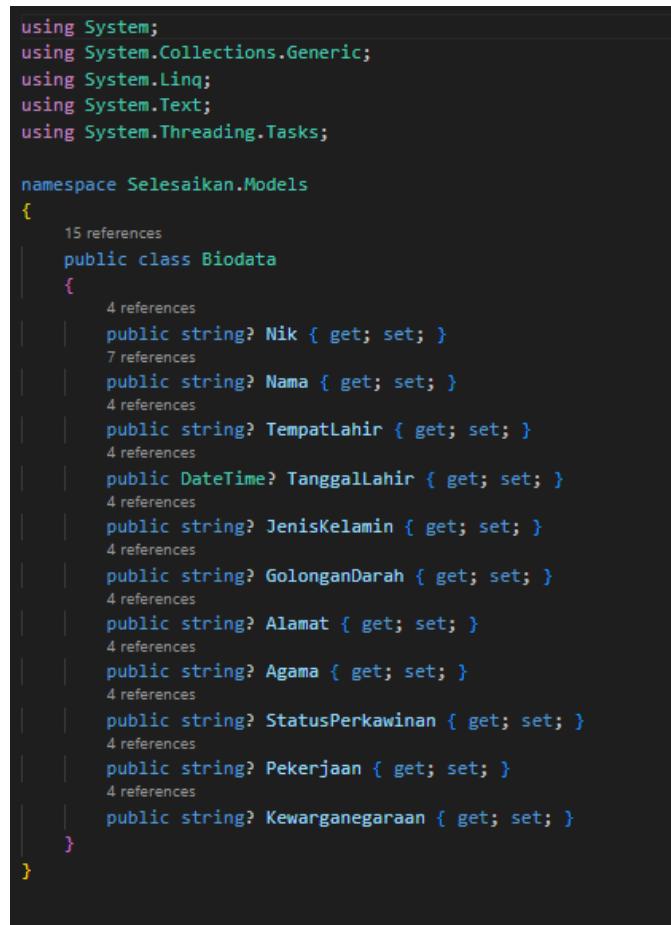
Folder ini hanya bertujuan untuk menyimpan gambar dan *asset* yang digunakan untuk membangun GUI dari aplikasi ini. Gambar-gambar yang terdapat pada folder ini dapat dilihat pada lampiran gambar berikut.



**Gambar 4.1.6.** Cuplikan gambar-gambar yang terdapat pada folder Images

#### e. Folder *Models*

Folder ini bertujuan untuk menyimpan model-model setiap entitas yang terdapat pada basis data yang telah dibangun sebelumnya. Terdapat dua jenis model karena terdapat dua tabel relasional yang digunakan dalam aplikasi ini. Dua jenis model tersebut disimpan dalam file yang berbeda, yaitu *Biodata.cs* dan *SidikJari.cs*. Terdapat juga file *AppConfig.cs* untuk menyimpan config yang diterapkan untuk menyimpan konfigurasi aplikasi. Cuplikan salah satu model yang disusun dapat dilihat pada gambar yang terlampir berikut.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Selesaikan.Models
{
    15 references
    public class Biodata
    {
        4 references
        public string? Nik { get; set; }
        7 references
        public string? Nama { get; set; }
        4 references
        public string? TempatLahir { get; set; }
        4 references
        public DateTime? TanggalLahir { get; set; }
        4 references
        public string? JenisKelamin { get; set; }
        4 references
        public string? GolonganDarah { get; set; }
        4 references
        public string? Alamat { get; set; }
        4 references
        public string? Agama { get; set; }
        4 references
        public string? StatusPerkawinan { get; set; }
        4 references
        public string? Pekerjaan { get; set; }
        4 references
        public string? Kewarganegaraan { get; set; }
    }
}
```

**Gambar 4.1.7.** Cuplikan model basis data Biodata

f. Folder *test*

Folder ini berisi gambar-gambar sidik jari oleh orang-orang yang terdaftar dalam basis data. Folder ini tidak tersedia dalam *repository* dan *di-ignore* sehingga pengisian folder ini dengan *dataset* sidik jari orang-orang secara manual oleh pengguna aplikasi ini.

Lampiran *dataset* sidik jari yang dapat digunakan dapat diunduh dari lampiran laporan ini.

#### g. Folder *Utils*

Folder ini berisi file *Encrypt.cs* yang merupakan file yang dibuat dengan tujuan untuk menyimpan fungsi dekripsi dan enkripsi yang digunakan untuk menangani kasus sekuritas tambahan terhadap informasi pemilik sidik jari yang terdapat pada basis Biodata dan *Utils.cs* yang berisi fungsi-fungsi pembantu yang digunakan untuk membantu pembangunan fungsi-fungsi atau algoritma lain yang terdapat pada aplikasi ini, seperti fungsi untuk melakukan konversi dari suatu bentuk gambar menjadi bentuk gambar lain dengan fungsionalitas yang berbeda. Berikut merupakan cuplikan untuk kode yang terdapat pada kedua file tersebut.

```
public class Blowfish
{
    // P-array nyimpan 18 subkey hasil ekspansi
    9 references
    private uint[] P;
    // S-boxes nyimpan value sembarang, untuk membuat data non-linear sehingga lebih aman
    6 references
    private uint[,] S;
    6 references
    private const int BlockSize = 8;

    3 references
    public Blowfish(byte[] key)
    {
        |   SetupKey(key);
    }

    1 reference
    private void SetupKey(byte[] key)
    {
        |
        P = new uint[18];
        S = new uint[4, 256];

        // P-array bakal dininisialisasi dengan nilai i, trus bakal diXOR-in sama key spt di bawah
        // Untuk s-boxes, dininisialisasi nilai random, disini dibuat seperti dibawah.
        for (int i = 0; i < 18; i++) P[i] = (uint)i;
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 256; j++)
                S[i, j] = (uint)(i * 256 + j);

        int keyIndex = 0;
        for (int i = 0; i < 18; i++)
        {
            // Mencari nilai untuk diXORin sama nilai p-array sekarang
            // Penambahan key dilakukan berkelanjutan antara subkey, jadi misal
            // Untuk subkey-0 keyIndex berakhir di 2, maka subkey-1 nilai data nya mulai dari indexkey-3, dst
            uint data = 0x00000000;
            for (int j = 0; j < 4; j++)
            {
                data = (data << 8) | key[keyIndex];
                keyIndex++;
                if (keyIndex >= key.Length)
                    keyIndex = 0;
            }
            P[i] ^= data;
        }
    }
}
```

**Gambar 4.1.8.** Cuplikan kode kelas Blowfish pada file *Encrypt.cs*

```

public class Utils
{
    2 references
    public static Bitmap BitmapImage2Bitmap(BitmapImage bitmapImage)
    {
        if (bitmapImage.StreamSource == null || !bitmapImage.StreamSource.CanRead)
        {
            throw new ArgumentException("StreamSource is either null or cannot be read");
        }

        // Copy the stream to keep the original stream intact
        MemoryStream outStream = new MemoryStream();
        bitmapImage.StreamSource.Seek(0, SeekOrigin.Begin); // Ensure we start from the beginning
        bitmapImage.StreamSource.CopyTo(outStream);
        outStream.Seek(0, SeekOrigin.Begin); // Reset the position to the beginning

        // Create bitmap from copied stream
        return new Bitmap(outStream);
    }

    // Fungsi ConvertToGrayscale untuk mengubah gambar bitmap original menjadi bitmap grayscale
    2 references
    public static Bitmap ConvertToGrayscale(Bitmap original)
    {
        Bitmap grayscale = new Bitmap(original.Width, original.Height);

        for (int i = 0; i < original.Width; i++)
        {
            for (int j = 0; j < original.Height; j++)
            {
                Color originalColor = original.GetPixel(i, j);
                int grayScale =
                    (int)((originalColor.R * 0.3) + (originalColor.G * 0.59) + (originalColor.B * 0.11));
                Color grayColor = Color.FromArgb(grayScale, grayScale, grayScale);
                grayscale.SetPixel(i, j, grayColor);
            }
        }
    }
}

```

**Gambar 4.1.9.** Cuplikan kode kelas Utils pada file *Utils.cs*

### Penjelasan Bonus Encrypt

Pada projek ini, kami menggunakan enkripsi *Blowfish*, yang dimana merupakan metode enkripsi umum yang simetrik dan beroperasi pada data 64 bit. Berikut adalah tahap-tahapan enkripsi (tahapan lebih jelas terdapat pada kode).

1. Ekspansi Kunci

Kunci simetrik kemudian diekspansi menjadi 18 subkey. Subkey tersebut dilakukan dengan melakukan operasi XOR dari indeks key terhadap nilai key utama.

2. Inisialisasi nilai S-Box

Nilai s-box yang berukuran  $4 \times 256$  diinisialisasi dengan

$$S[i, j] = (\text{uint})(i * 256 + j),$$

Dimana  $i$  berupa variabel iterator row s-box (0-3) dan  $j$  berupa variabel iterator column s-box (0-255).

3. Penambahan Padding pada Data untuk Enkripsi

Data yang akan dienkripsi, akan terlebih dahulu dilakukan padding (tambahan bit) sehingga menjadi kelipatan 64 bit. Hal ini dikarenakan enkripsi BlowFish yang beroperasi pada data berukuran 64 bit

4. Enkripsi

Data yang telah menjadi kelipatan 64 bit kemudian dienkripsi dengan melakukan operasi XOR, swap, dan tambah dengan tiap subkeys (menaik) dan tiap nilai S-Box. Hal ini agar data menjadi tidak linear dan tidak traceable, terutama dengan penambahan S-Box yang mengacak-acak nilai data.

#### 5. Pengurangan Padding pada Data untuk Decrypt

Data yang telah ditambahkan padding sebelumnya ketika dienkripsi, akan kemudian dibuang padding tambahan tersebut agar bisa didecrypt. Data tentang jumlah padding yang ditambahkan disimpan pada indeks byte padding yang telah ditambahkan sebelumnya

#### 6. Decrypt

Data yang telah dibuang paddingnya, kemudian dilakukan operasi decrypt. Operasi decrypt mirip-mirip dengan operasi enkripsi. Perbedaannya hanya pada arah iterasi subkeys yang menurun. Pada enkripsi, iterasi subkeys dimulai dari subkey pertama hingga subkey ke-18. Hal ini kebalikannya pada decrypt, dimana iterasi subkey dimulai dari subkey ke-18 hingga subkey pertama.

#### h. File *docker-compose.yaml*

File *docker-compose.yaml* adalah file konfigurasi yang digunakan oleh *Docker Compose*, sebuah alat untuk mendefinisikan dan mengelola aplikasi *multi-container Docker*. *Docker Compose* memungkinkan penggunaannya untuk mengatur aplikasi dengan layanan-layanan yang terpisah, mendefinisikan hubungan antar layanan, dan mengelola keseluruhan *lifecycle* aplikasi secara efisien.

```

version: "3.8"
services:
  adminer:
    image: adminer
    restart: always
    ports:
      - "8080:8080"
  networks:
    selesaikan-network:
      aliases:
        - adminer.selesaikan.local

  mysql-service:
    image: mysql:latest
    container_name: mysql-service
    hostname: mysql-service

```

**Gambar 4.1.10.** Cuplikan isi dari file *docker-compose.yaml*

i. selesaikan-db.sql

File selesaikan-db.sql berisi kode-kode sql yang dapat di-*execute* sehingga dapat melakukan pembangunan basis data pada *local database* pengguna aplikasi ini. Pemasukan informasi-informasi data biodata dan sidik jari pada tabel dilakukan pada file SQL ini.

```

SET NAMES utf8;
SET time_zone = '+00:00';
SET foreign_key_checks = 0;
SET sql_mode = 'NO_AUTO_VALUE_ON_ZERO';

SET NAMES utf8mb4;

DROP TABLE IF EXISTS `biodata`;
CREATE TABLE `biodata` (
  `NIK` varchar(36) NOT NULL,
  `nama` varchar(512) DEFAULT NULL,
  `tempat_lahir` varchar(512) DEFAULT NULL,
  `tanggal_lahir` date DEFAULT NULL,
  `jenis_kelamin` enum('Laki-Laki','Perempuan') DEFAULT NULL,
  `golongan_darah` varchar(36) DEFAULT NULL,
  `alamat` varchar(512) DEFAULT NULL,
  `agama` varchar(256) DEFAULT NULL,
  `status_perkawinan` enum('Belum Menikah','Menikah','Cerai') DEFAULT NULL,
  `pekerjaan` varchar(512) DEFAULT NULL,
  `kewarganegaraan` varchar(512) DEFAULT NULL,
  PRIMARY KEY (`NIK`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO `biodata`(`NIK`, `nama`, `tempat_lahir`, `tanggal_lahir`, `jenis_kelamin`, `golongan_darah`, `alamat`, `agama`, `status_perkawinan`, `pekerjaan`, `kewarganegaraan`)
VALUES ('3174010101597346', 'CH15 F7r1 WnRn', 'Surabaya', '1958-01-01', 'Laki-Laki', 'AB', 'Gg. Jakarta Raya Blok D No. 15', 'Islam', 'Belum Menikah', 'PNS', 'Indonesia'), ('3174010103847590', 'MOR Krt SRYm', 'Binjai', '1984-03-01', 'Laki-Laki', 'A', 'Gang Moch. Syaikhul Huda No. 1', 'Islam', 'Belum Menikah', 'PNS', 'Indonesia'), ('3174010108788473', 'Iys Dz 7Fm SRGH', 'Salatiga', '1978-08-01', 'Laki-Laki', 'O', 'Jl. Merdeka Selatan No. 12', 'Islam', 'Belum Menikah', 'PNS', 'Indonesia'), ('3174010204853361', 'WV VICKY 9TR RYN7', 'Pontianak', '1985-04-02', 'Laki-Laki', 'B', 'Jl. Prof. DR. Soekarno No. 12', 'Islam', 'Belum Menikah', 'PNS', 'Indonesia'), ('3174010207789677', 'Jig PrMD', 'Jayapura', '1978-07-02', 'Laki-Laki', 'O', 'Jl. Astana Anya No. 12', 'Islam', 'Belum Menikah', 'PNS', 'Indonesia'), ('3174010301892492', 'VckY Shn prM40', 'Tanjungpinang', '1989-01-03', 'Laki-Laki', 'B', 'Jl. Sultan Iskandar No. 12', 'Islam', 'Belum Menikah', 'PNS', 'Indonesia'), ('3174010312839798', 'Bng HR7 hRSWn B JNC', 'Gorontalo', '1983-12-03', 'Laki-Laki', 'A', 'Ge')

```

**Gambar 4.1.11.** Cuplikan kode SQL untuk membangun basis data aplikasi ini

j. MainWindow.xaml.cs dan MainWindow.xaml

Dalam pengembangan aplikasi menggunakan *Windows Presentation Foundation* (WPF) di .NET, file MainWindow.xaml dan file *code-behind* MainWindow.xaml.cs adalah dua komponen penting yang bekerja bersama untuk membangun antarmuka pengguna dan logika aplikasi.

File MainWindow.xaml adalah file XAML (*eXtensible Application Markup Language*) yang mendefinisikan struktur dan tampilan antarmuka pengguna (UI) dari jendela utama aplikasi. XAML memungkinkan pengembang untuk mendefinisikan elemen-elemen UI seperti tombol, teks, kotak input, dan layout secara deklaratif.

File MainWindow.xaml.cs adalah file C# yang berperan sebagai code-behind untuk MainWindow.xaml. File ini mengandung logika pemrograman yang mengendalikan perilaku UI yang didefinisikan di file XAML. Ini termasuk event handlers untuk menangani aksi pengguna seperti klik tombol, input gambar dari *local directory PC* pengguna, dan interaksi lainnya.

File MainWindow.xaml.cs berisi fungsi-fungsi dan aksi yang dilakukan ketika pengguna menekan sebuah tombol, seperti tombol *Search*. Pada file ini pula semua kode-kode algoritma dan logika yang pernah dibuat pada file / kelas lainnya, digabung menjadi satu hingga membentuk sebuah aplikasi yang interaktif dan memenuhi fungsionalitas. Ketika pengguna menekan tombol *search*, aplikasi ini akan menjalankan program dan melakukan penyamaan *pattern matching* sehingga mengeluarkan sebuah *output* akhir berupa sidik jari yang paling *match* dan biodata daripada pemilik sidik jari yang *match* tersebut.

```
<Image Panel.ZIndex="1" HorizontalAlignment="Center" VerticalAlignment="Center" Height="320" Margin="-15,-18,0,0" Width="1047" Source="Images/background.jpg" Dislocked="True"/>
<Label FontSize="20pt" Content="Biometrik - Selesaikan" Foreground="Aure" HorizontalAlignment="Center" FontFamily="Comic Sans MS"/>
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Top" Width="880" Height="364" Margin="0,0,0,0"/>
    <WrapPanel HorizontalAlignment="Center" VerticalAlignment="Top" Width="880" Height="364" Margin="0,0,0,0"/>
        <Image Margin="10,0,0,0" x:Name="insertImage" Source="Images/insertImage.png" Height="250" Width="250" Stretch="UniformToFill" HorizontalAlignment="Center" VerticalAlignment="Top"/>
        <Image Margin="10,0,0,0" x:Name="outputImage" Source="Images/insertImage.png" Height="250" Width="250" Stretch="UniformToFill" HorizontalAlignment="Center" VerticalAlignment="Top"/>
    <StackPanel Margin="10,0,20,0" Width="230" Height="250" Background="#80327689" HorizontalAlignment="Center">
        <StackPanel Orientation="Horizontal">
            <Label x:Name="TitleUser" Content="Biodata Pengguna" Foreground="White" FontSize="16" FontWeight="Bold" HorizontalContentAlignment="Center"/>
            <StackPanel Orientation="Vertical">
                <Label Margin="5,-5,0,0" x:Name="NIK" Content="NIK" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
                <Label Margin="5,-5,0,0" x:Name="NIKUser" Content="NIK" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
            </StackPanel>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <Label Margin="5,-5,0,0" x:Name="Nama" Content="Nama" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
            <Label Margin="5,-5,0,0" x:Name="NamaUser" Content="Nama" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <Label Margin="5,-5,0,0" x:Name="TempatLahir" Content="Tempat Lahir" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
            <Label Margin="5,-5,0,0" x:Name="TempatLahirUser" Content="Tempat Lahir" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <Label Margin="5,-5,0,0" x:Name="TanggalLahir" Content="Tanggal Lahir" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
            <Label Margin="5,-5,0,0" x:Name="TanggalLahirUser" Content="Tanggal Lahir" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
        </StackPanel>
        <StackPanel Orientation="Horizontal">
            <Label Margin="5,-5,0,0" x:Name="JenisKelamin" Content="Jenis Kelamin" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
            <Label Margin="5,-5,0,0" x:Name="JenisKelaminUser" Content="Jenis Kelamin" Foreground="White" FontSize="18" FontWeight="Bold" HorizontalContentAlignment="Left"/>
        </StackPanel>
    </StackPanel>

```

**Gambar 4.1.12.** Cuplikan kode pada file MainWindow.xaml

```

public MainWindow()
{
    InitializeComponent();
    SetInitialImage();
    currentActiveAlgorithm = "KMP";
    _database = new Database.Database();
    UpdateButtonColors();
    resultSidikJari = new SidikJari();
    resultBiodata = new Biodata();
    entryImage = new BitmapImage();

    this.stopwatch = new Stopwatch();

    AppConfig loadedConfig = Config.Config.LoadConfiguration("config.json");
    if (!loadedConfig.IsEncrypted)
    {
        List<Biodata> allBiodata = _database.GetAllBiodata();
        _database.TruncateBiodata();
        _database.SaveAllBiodata(allBiodata);
        loadedConfig.IsEncrypted = true;
        Config.Config.SaveConfiguration(loadedConfig, "config.json");
    }
}

public void UpdateResultBiodata()
{
    AppConfig loadedConfig = Config.Config.LoadConfiguration("config.json");
    byte[] key = Encoding.UTF8.GetBytes(loadedConfig.Key);
    Blowfish blowfish = new Blowfish(key);

    NamaUser.Content = resultSidikJari.Nama;
    NikUser.Content = blowfish.Decrypt(resultBiodata.Nik);
    TempatLahirUser.Content = blowfish.Decrypt(resultBiodata.TempatLahir);
    TanggalLahirUser.Content = resultBiodata.TanggalLahir;
    JenisKelaminUser.Content = resultBiodata.JenisKelamin;
}

```

**Gambar 4.1.13.** Cuplikan kode pada file MainWindow.xaml.cs

## 4.2. Penjelasan Tata Cara Penggunaan Program

Terdapat beberapa langkah-langkah yang harus dilakukan untuk menggunakan program ini. Namun, sebelum itu, terdapat beberapa fitur yang disediakan oleh program, yaitu pencarian sidik jari yang paling *matching* dengan dua jenis algoritma, yaitu algoritma KMP (*Knuth-Morris-Pratt*) dan algoritma BM (*Boyer-Moore*). Selain itu, terdapat pula fitur tambahan, yaitu aplikasi dapat membaca nama yang terkorupsi database sehingga format nama menjadi dalam format bahasa Alay. Terakhir, terdapat juga fitur tambahan lainnya, yaitu perlakuan enkripsi terhadap seluruh informasi biodata yang terdapat pada basis data Biodata bersama dengan nama pemilik sebuah sidik jari. Enkripsi ini dilakukan dengan menggunakan algoritma *blowfish*. Dengan adanya fitur-fitur tambahan ini, maka aplikasi ini menjadi lebih aman yang mana aplikasi dapat membaca data yang terkorupsi serta data-data yang terdapat pada basis data menjadi lebih aman dan susah untuk dibobol oleh orang tak bertanggung jawab tanpa memiliki kunci dekripsi.

Langkah-langkah untuk menjalankan aplikasi pencocokan sidik jari, yaitu sebagai berikut.

1. Pertama, pastikan memiliki kode daripada aplikasi ini, dengan melakukan *cloning* dari *github* yang terdapat pada lampiran laporan ini. Dapat dilakukan dengan command sebagai berikut pada *terminal*, seperti berikut.

```
git clone https://github.com/mybajwk/Tubes3_SeleSAikan.git
```

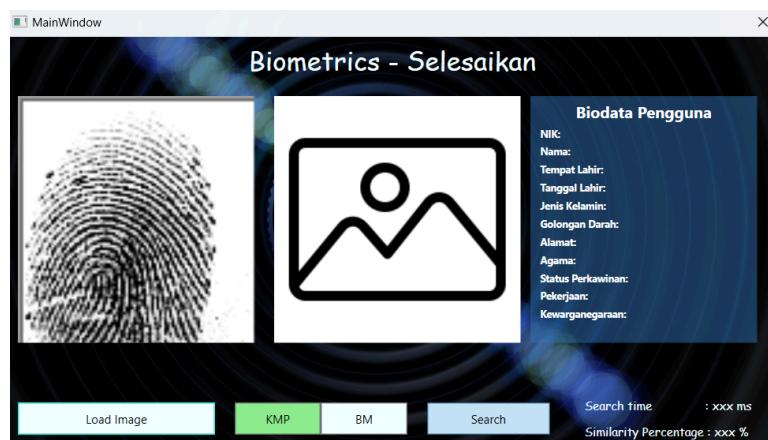
2. Kemudian, lakukan *docker compose up* pada *root folder* yang mengandung file docker.compose.yaml. Pastikan *docker desktop* sudah dihidupkan terlebih dahulu sebelum menjalankan aplikasi ini.
3. Setelah itu, pastikan *docker* sudah berjalan dengan baik dengan memastikan pada *localhost:8080* terdapat Adminer yang telah aktif untuk memudahkan perlakukan *import SQL* pada *database*. Dengan adanya Adminer ini, pengguna akan lebih mudah berinteraksi dengan basis data.
4. Untuk mengakses Adminer, akan diminta data-data untuk verifikasi untuk mengakses basis data yang hendak diakses, dengan *credentials* sebagai berikut.
  - a. System : MySQL
  - b. Server : mysql-service.selesaikan.local
  - c. Username : selesaikan
  - d. Password : selesaikan
  - e. Database : selesaikan-db
5. Pada Adminer yang terdapat pada *localhost:8080*, klik tombol “import” dan pilih file selesaikan-db.sql dan kemudian tombol “execute”. Maka, dengan itu maka semua basis data yang dibutuhkan aplikasi sudah ter-setup dengan baik beserta dengan data-data *dummy* untuk *relative path* gambar sidik jari beserta dengan nama orang pemiliknya.
6. Lakukan run aplikasi dengan melakukan *run and debug* pada *code editor* kesayangan ada. Aplikasi ini telah dicoba di-run dengan menggunakan *code editor* Visual Studio Code, Microsoft Visual Studio, Rider.
7. Pengguna dapat memilih terlebih dahulu dengan menekan tombol “KMP” atau “BM” untuk menemukan jenis algoritma apa yang hendak yang digunakan untuk melakukan pencocokan sidik jari.
8. Klik tombol “Load Image” untuk memilih foto yang hendak dibandingkan dengan sidik jari yang terdapat pada basis data. Berdasarkan gambar sidik jari yang di-load oleh

pengguna, aplikasi akan mencari sidik jari yang paling *matching* berdasarkan algoritma yang dipilih.

9. Pada akhirnya, setelah seluruh langkah sebelumnya dilakukan secara prosedural, maka akan muncul *output* gambar sidik jari yang *match* dengan gambar sidik jari yang dimasukkan oleh pengguna. Selain itu, juga terdapat informasi pemilik sidik jari yang *match* tersebut di sampingnya. Juga terdapat informasi lama pencarian sidik jari tersebut dalam format ms (milisekon) jika waktu eksekusi  $< 1000\text{ms}$ , dan format s (sekon) jika hasil dalam  $>1000\text{ms}$ . Kemudian, terdapat juga informasi *similarity percentage* (persentase kemiripan) yang didapatkan dari hasil *hamming distance* dibagi dengan panjang string yang diambil dari sebuah sidik jari.

### 4.3. Hasil Pengujian

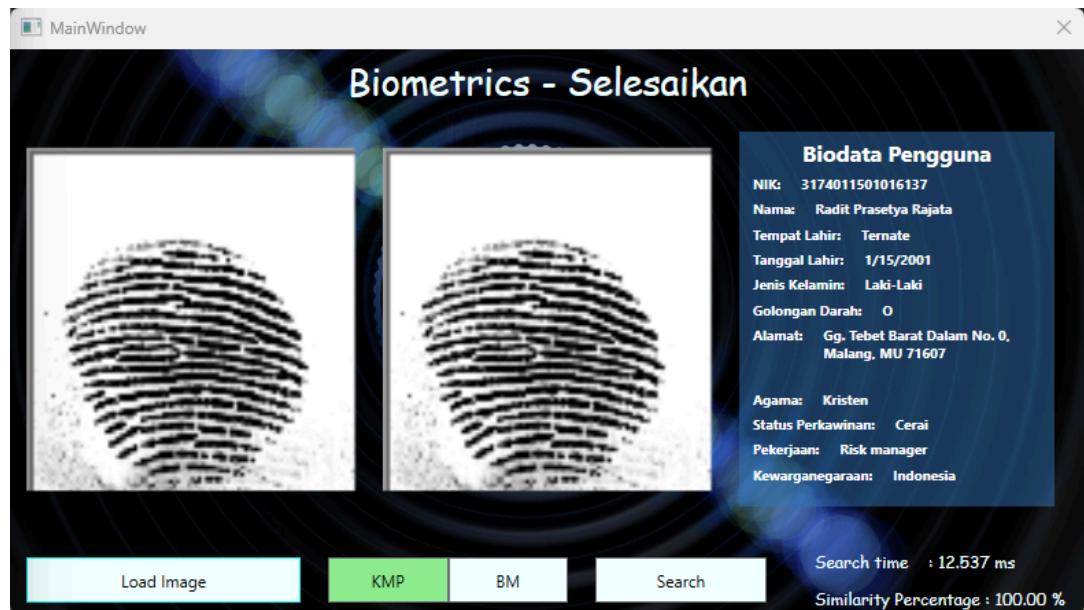
Sebelum melakukan pengujian, tampilan GUI aplikasi dapat dilihat pada lampiran berikut.



**Gambar 4.3.1.** Cuplikan GUI aplikasi sebelum pencarian

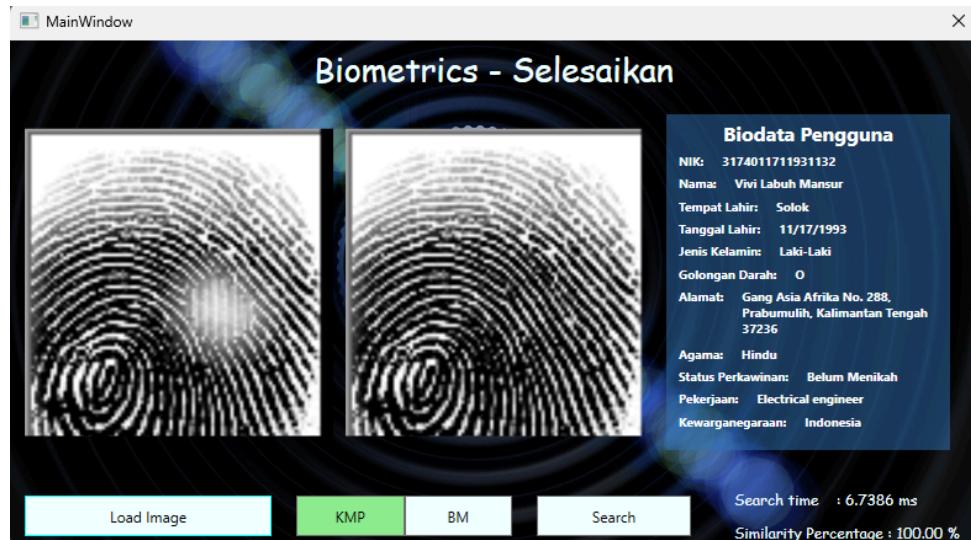
#### 4.3.1. Pengujian menggunakan Algoritma KMP (Knuth-Morris-Pratt)

##### a. Pengujian I (Real)



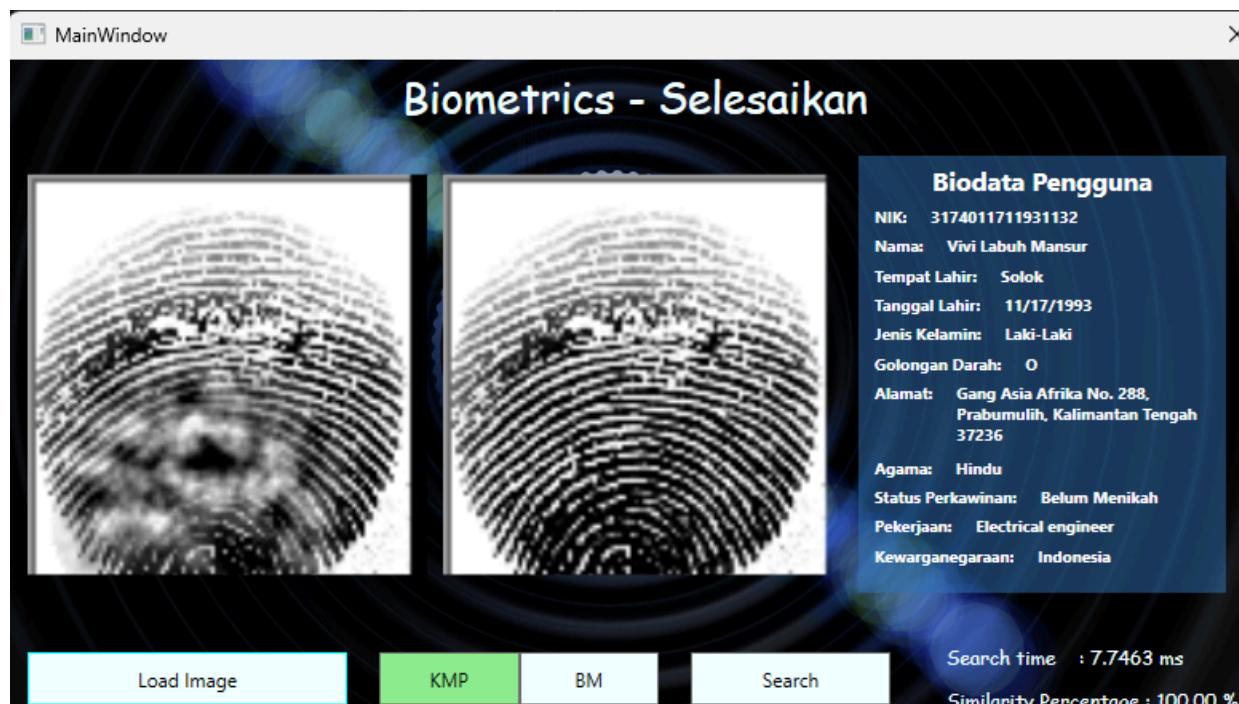
*Gambar 4.3.1.1.* Pengujian I Pencocokan Sidik Jari Algoritma KMP

##### b. Pengujian II (Altered-Low)



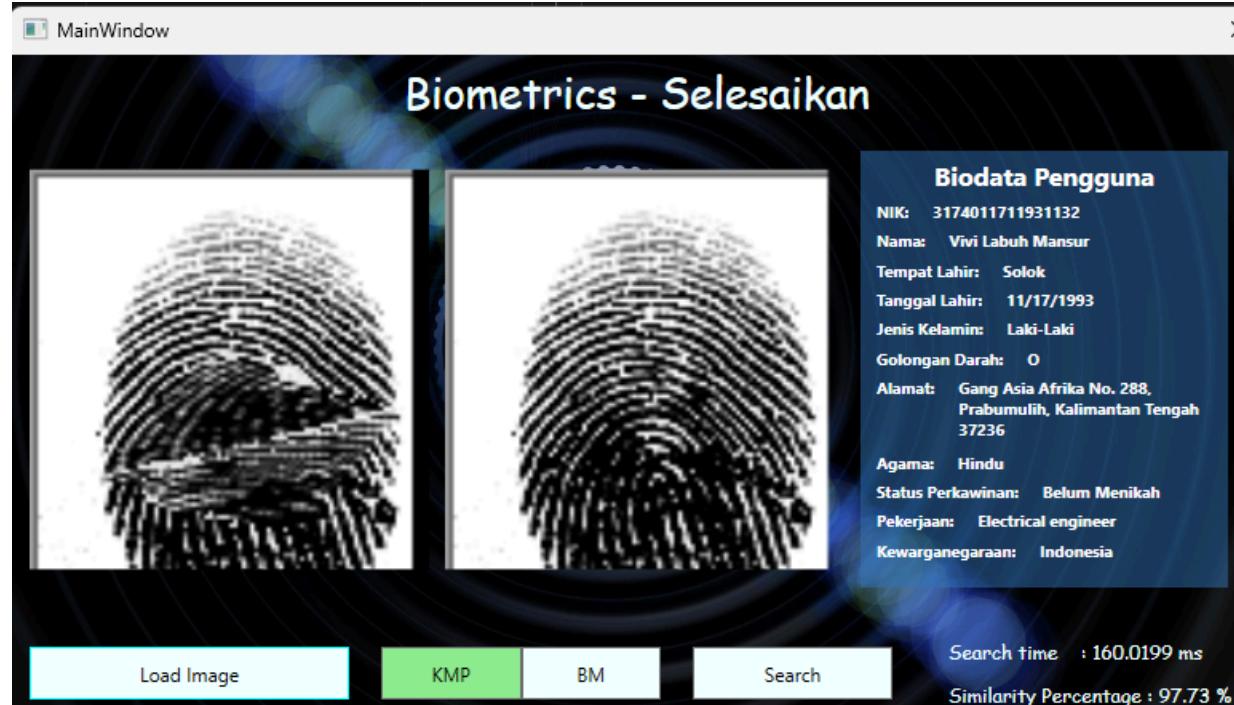
**Gambar 4.3.1.2.** Pengujian II Pencocokan Sidik Jari Algoritma KMP

c. Pengujian III (Altered-Mid)



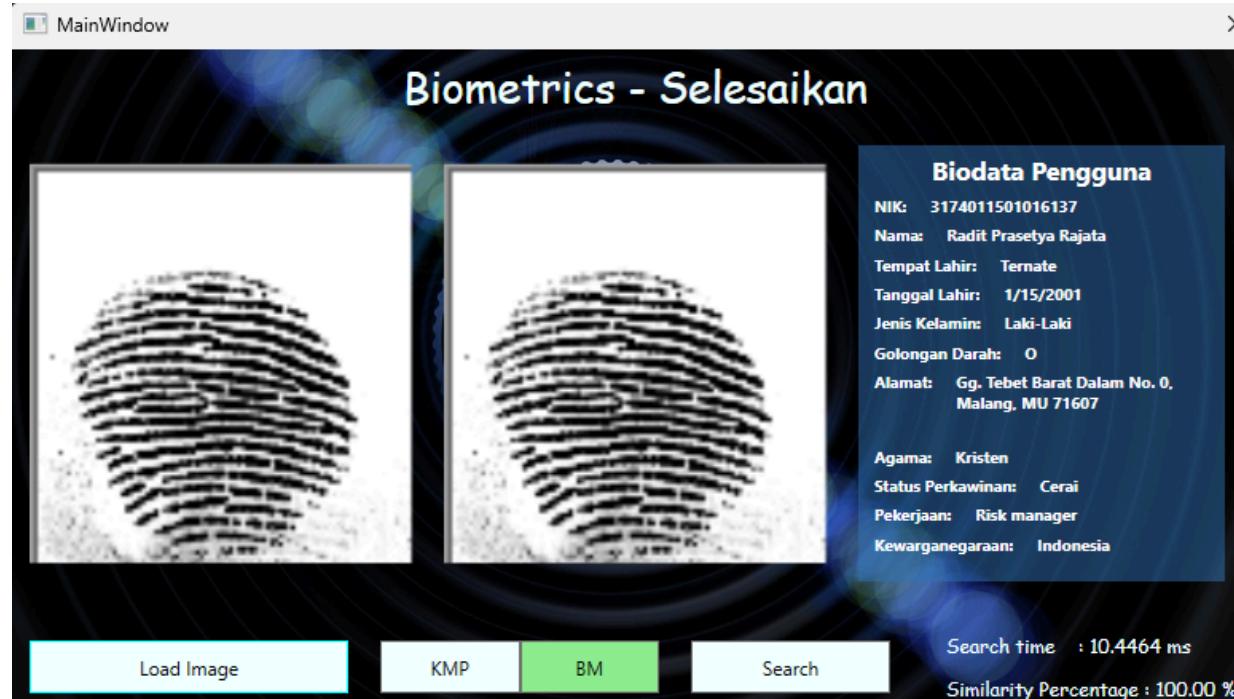
**Gambar 4.3.1.3.** Pengujian II Pencocokan Sidik Jari Algoritma KMP

d. Pengujian IV (Altered-High)



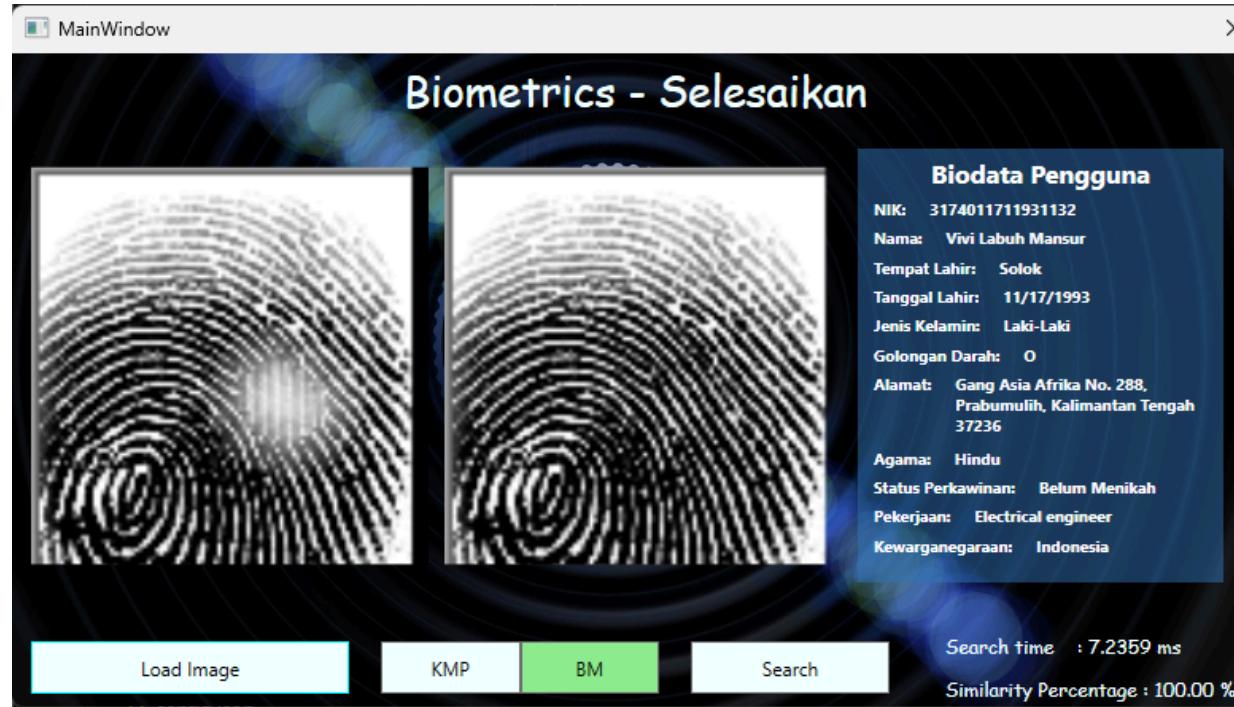
#### 4.3.2. Pengujian menggunakan Algoritma BM (Boyer Moore)

##### a. Pengujian I (Real)



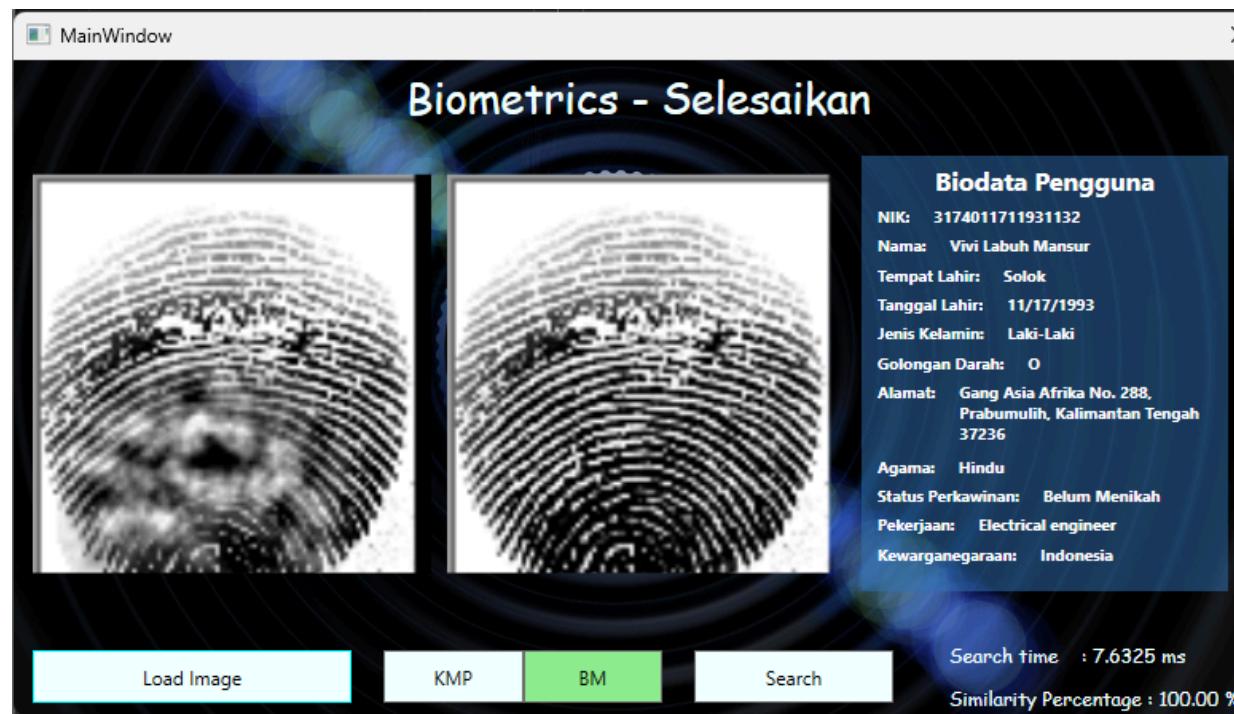
*Gambar 4.3.2.1. Pengujian I Pencocokan Sidik Jari Algoritma BM*

##### b. Pengujian II (Altered-Low)



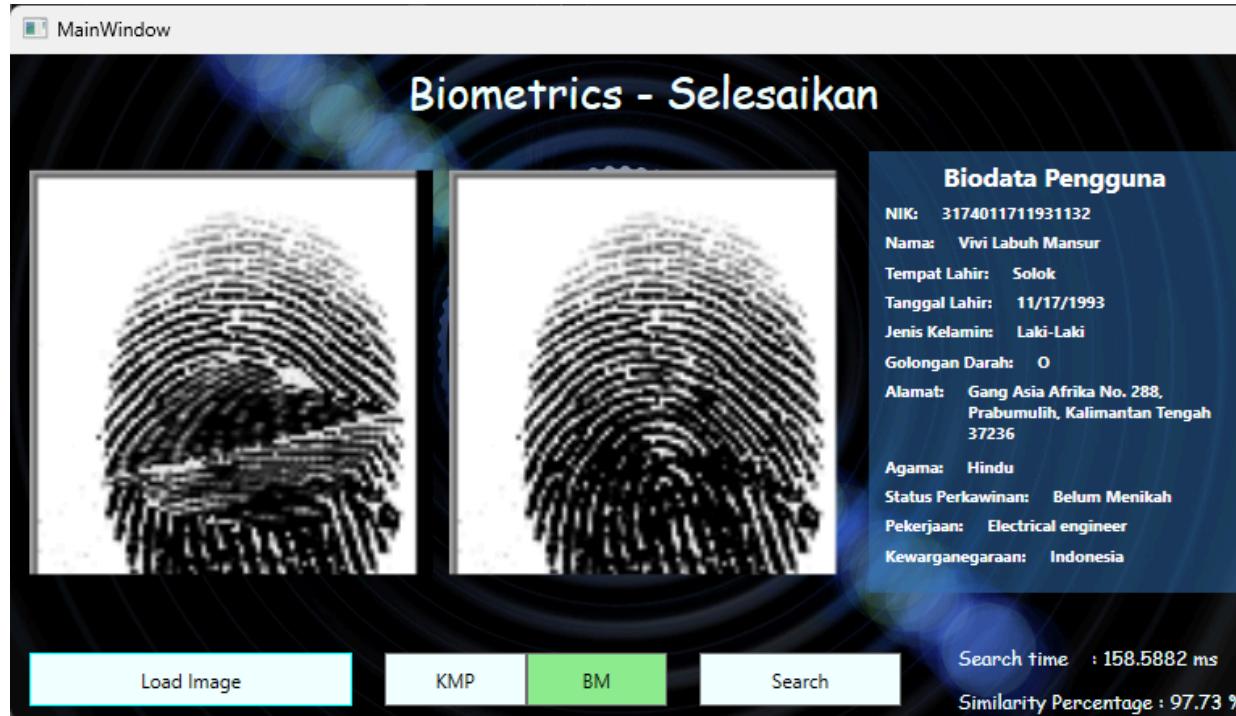
**Gambar 4.3.2.2.** Pengujian II Pencocokan Sidik Jari Algoritma BM

- c. Pengujian III (Altered-Mid)



**Gambar 4.3.2.3.** Pengujian III Pencocokan Sidik Jari Algoritma BM

d. Pengujian IV (Altered-High)



#### 4.3.3. Pencarian Sidik Jari dengan Pemilik Sidik Jari dengan Nama Terkorupsi

Untuk melakukan konversi nama terkorupsi (nama alay) menjadi nama yang dapat dibaca oleh manusia (*readable*), tentunya diperlukan fungsi untuk melakukan konversi tersebut. Untuk mencapai fungsi tersebut, dibutuhkan bantuan *regular expression (regex)*. Bukti atau hasil pengujian atas keberhasilan fitur ini dapat terlihat pada **Bagian 4.3.1.** dan **Bagian 4.3.2.**

<input type="checkbox"/> Modify	NIK	nama	tempat_lahir	tanggal_lahir	jenis_kelamin	golongan_darah	
<input type="checkbox"/> edit	31740101010587346	cHl5 f7r1 WnRn	Surabaya	1958-01-01	Laki-Laki	AB	Gg. Jakarta No. 128, Salatiga
<input type="checkbox"/> edit	3174010103847590	MJR Krt SRYTm	Binjai	1984-03-01	Laki-Laki	A	Gang Moch. Toha No. 10, Por
<input type="checkbox"/> edit	3174010108788473	lyS DW 7Fn SR6H	Salatiga	1978-08-01	Laki-Laki	O	Jl. Merdeka No. 017, Surabaya
<input type="checkbox"/> edit	3174010204853361	Mr V1CKy 9TR RYN7	Pontianak	1985-04-02	Laki-Laki	B	Jalan Suryakencana No. 33, J
<input type="checkbox"/> edit	3174010207789677	jg PrMD	Jayapura	1978-07-02	Laki-Laki	O	Jl. Astana Anyar No. 7, Payal
<input type="checkbox"/> edit	3174010301892492	vCKy Shn prM4D	Tanjungpinang	1989-01-03	Laki-Laki	B	Jalan Setiabudhi No. 860, La
<input type="checkbox"/> edit	3174010312839298	BnR HR7 hRSNt0 JNr	GORONTALO	1983-12-03	Laki-Laki	A	Gg. Ahmad Yani No. 565, Sib
<input type="checkbox"/> edit	3174010402894265	bIND 3d IN69nG SMNjN7K	Solo	1989-02-04	Laki-Laki	A	Gg. Rajiman No. 977, Siboli
<input type="checkbox"/> edit	3174010504888914	8d PNgst 5t	Sukabumi	1988-04-05	Laki-Laki	AB	Jalan Ir. H. Djunda No. 7, B
<input type="checkbox"/> edit	31740105059918449	LGw K3nZ3 KJN n6GrWn	Cilegon	1991-09-05	Laki-Laki	AB	Gang Gegerkalong Hilir No. C
<input type="checkbox"/> edit	3174010608603453	Cmt1 JN TM	Banda Aceh	1960-08-06	Laki-Laki	A	Gg. Suryakencana No. 80, M
<input type="checkbox"/> edit	3174010610845989	r7h KCn6 nM9	Kota Administrasi Jakarta Pusat	1984-10-06	Laki-Laki	A	Gg. Ahmad Dahlan No. 532,
<input type="checkbox"/> edit	3174010611019331	vN kNt l4mp80Ln	Sawahlunto	2001-11-06	Laki-Laki	A	Gang Kiacondono No. 6, M
<input type="checkbox"/> edit	3174010611705346	Hld mnGNSN6	Pasuruan	1970-11-06	Laki-Laki	O	Gang M.H Thamrin No. 960,
<input type="checkbox"/> edit	3174010804628177	JsMn wRD mPr55T	Surabaya	1962-04-08	Laki-Laki	AB	Gang Medokan Ayu No. 990,
<input type="checkbox"/> edit	3174010806527304	b7n 8Nr pdjst7	Bitung	1952-06-08	Laki-Laki	O	Gg. Veteran No. 599, Sabang
<input type="checkbox"/> edit	3174010901036256	cNDY prb SR3GR	Tidore Kepulauan	2003-01-09	Laki-Laki	AB	Gang Indragiri No. 15, Banj
...							

**Gambar 4.3.3.1.** Cuplikan pertama Basis Data Biodata yang memiliki nama pemilik terkorupsi

<input type="checkbox"/> edit	3174011604455909	knz cndrkNt pNd trHRRN	Kota Administrasi Jakarta Barat	1945-04-16	Laki-Laki	B	Gg. Jamika No. 355, Kotamobagu,
<input type="checkbox"/> edit	3174011607469197	PN11 Rdn7	Banjarmasin	1946-07-16	Laki-Laki	A	Jl. Surapati No. 2, Subulusalam, S
<input type="checkbox"/> edit	3174011607873835	WRS7 4Rs W8W	Bandung	1987-07-16	Laki-Laki	AB	Gg. Asia Afrika No. 9, Kendari, Mal
<input type="checkbox"/> edit	3174011608039997	pDm HMR ml TM	Banjarbaru	2003-08-16	Laki-Laki	A	Jalan R.E Martadinata No. 8, Goron
<input type="checkbox"/> edit	3174011701886246	vR Sm8L0N	Kota Administrasi Jakarta Barat	1988-01-17	Laki-Laki	B	Jalan Jamika No. 76, Kediri, GO 67
<input type="checkbox"/> edit	3174011705571165	MMpn ftr4 ZlkrNn	Medan	1957-05-17	Laki-Laki	AB	Jalan Rajawali Timur No. 285, Banc
<input type="checkbox"/> edit	3174011711931132	V1V1 L4BH mN5	Solok	1993-11-17	Laki-Laki	O	Gang Asia Afrika No. 288, Prabum
<input type="checkbox"/> edit	3174011807737615	xNN rh4RJ LK n6rH0	Tarakan	1973-07-18	Laki-Laki	B	Gg. Monginsidi No. 51, Kota Admin
<input type="checkbox"/> edit	3174011810010424	D4cn nrDyNt	Kupang	2001-10-18	Laki-Laki	B	Jl. Pasirkoja No. 63, Kupang, AC 76
<input type="checkbox"/> edit	3174011901761008	srvd w64 m4rs m4RpN6	Balkpapan	1976-01-19	Laki-Laki	A	Jl. Cikutra Barat No. 93, Pekalonga
<input type="checkbox"/> edit	3174012006734099	d1Ms l s4R61H	Bogor	1973-06-20	Laki-Laki	O	Gg. Laswi No. 25, Kota Administras
<input type="checkbox"/> edit	3174012012923133	KcNG Ws	Sabang	1992-12-20	Laki-Laki	O	Gang Ahmad Dahlan No. 45, Banda
<input type="checkbox"/> edit	3174012104463636	9Mn 7m8l Hrj Wlndr	Jambi	1946-04-21	Laki-Laki	A	Jalan Waringin No. 931, Kediri, Pap
<input type="checkbox"/> edit	3174012108824828	RIN Ndh rrnt ktVN	Banjar	1982-08-21	Laki-Laki	AB	Jalan K.H. Wahid Hasyim No. 242,
<input type="checkbox"/> edit	3174012109024160	mplk Hrj DNNG s1mnjn74k	Meulaboh	2002-09-21	Laki-Laki	AB	Gg. Dipatiukur No. 829, Sibolga, At
<input type="checkbox"/> edit	3174012111166675	lvn lv4 ITpN	Depok	1946-11-21	Laki-Laki	AB	Gang Ciwastra No. 9, Pasuruan, Ri

**Gambar 4.3.3.2.** Cuplikan pertama Basis Data Biodata yang memiliki nama pemilik terkorupsi

#### 4.3.4. Pencarian dengan Basis Data Biodata yang Terenkripsi

Cuplikan basis data Biodata yang telah dienkripsi dengan menggunakan algoritma *Blowfish* dapat dilihat pada **Gambar 4.3.4.1.** dan **Gambar 4.3.4.2.** Pencarian pada **Bagian 4.3.1.** dan **Bagian 4.3.2.** telah menggunakan nama Alay dan kemudian dienkripsi beserta dengan informasi-informasi lainnya yang telah dienkripsi juga.

tempat_lahir	nama	NIK	Modify
THkNyMPxdxE4SyJ+xHFrM1xZMxmPkghfUoIDB8+VzQ=	Y11jabITII2DaCeM5DE5MEA==	/ghxOoEsMnRtDnc9Yik0c3AwSwVMFwpI	<input type="checkbox"/> edit
UU0iYRlwC+8P0QKQhgFRw==	NFAPYitzM3SIPkQKxRgFrw==	/ghxOoEsMnS7CHE1qCYydXawSwVMFwpI	<input type="checkbox"/> edit
/YtaLR7aSGxpKULTHEERg==	hnxLZZLTGaxGBN/C0Y2Djo+RQs5ewRG	/ghxOoEsMnTxAho5diY6d3AwSwVMFwpI	<input type="checkbox"/> edit
TUwmDOB6c4C=	qRgQVG9zCq5SpJgHV4Ylg+RarFGAVH	/Qh207MsNQXQcDHM6hy8c3AwSwVMFwpI	<input type="checkbox"/> edit
NVQ2Y6v/dSfaPpkULBXAERg==	hms0Y4fzamBIaEFP/3NQEa==	/QkQp1MsMrDno1X41cnAwSwVMFwpI	<input type="checkbox"/> edit
S1ckfZj+bCrUO0AOVXBjLg==	N24pdFvWWwwRwMsFTBckSA==	/wh2NE0tNXsgAHY/n06dHawSwVMFwpI	<input type="checkbox"/> edit
O0of3p6dBwMEsFTBckCSA==	RwwNLTSQUGBmrBc8n1pcAxawSwVMFwpI	/wh2NE0tNXt1DHM4Ei8yAxwSwVMFwpI	<input type="checkbox"/> edit
djTAjdB7c8l=	NBktYAreySCPEcJ2WZGKA==	+AhryPYOsMnq7CXU9yC06c3AwSwVMFwpI	<input type="checkbox"/> edit
xjpADkJ+dIE=	cQ6fVjN4/cndufXNvNdyw/RquiSwRG	+AhryPYOsMnq7DXEbg43cnAwSwVMFwpI	<input type="checkbox"/> edit
YV8zbM1+ZiEPoKAOIXRjLg==	71fQj7x5j1k4PUcJ3EtSDA==	+AhryPYOsMnR6Axo6/k1exAwSwVMFwpI	<input type="checkbox"/> edit
rIktan57byHSWDEMvndjLg==	Qwx2Q+NGUWCIPkQKxRgFrw==	+AhryPYOsMsNCAHE4dSg7c3AwSwVMFwpI	<input type="checkbox"/> edit
815BD993dJU=	0nd02QdsuImARGQs5hT83d/w+rQuAcRG	+AhryPYOsMnrgXYdHo47Cwyc3AwSwVMFwpI	<input type="checkbox"/> edit
xVQq1psb5QpuYB3htRQ==	104pVmfd6CP0ULXskERg==	+QhzNPKsNXRxDHo47Cwyc3AwSwVMFwpI	<input type="checkbox"/> edit
rEWubh8p+eyFAPkULPncERg==	+QhzNPKsNXTXHC3o+y81d3AwSwVMFwpI	+QhzNPKsNXTXHC3o+y81d3AwSwVMFwpI	<input type="checkbox"/> edit
tVsifyh6aSe9PkQKuxkFRw==	nVNO06fTaWdaTEEP7IVxL0==	+QhzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
ZkwxbC9+eyFwMEsFTBckCSA==	3AtjQStRNmCjYUEPgmyINw==	+wh1PudvNXTNAHA1gyw1cXAwSwVMFwpI	<input type="checkbox"/> edit
x1xjRl2Z2i+rTSjN293LHawSwVMFwpI	JldjabRNng00AOYyFFEg==	+wh1PudvNxtNHA1gyw1cXAwSwVMFwpI	<input type="checkbox"/> edit
pU0veJ7RbDwMEsFTBckCSA==	zEGxb7BQRhn6a0EPQkxQcw==	+whzNmlvNQXQnCn7s8yc3AwSwVMFwpI	<input type="checkbox"/> edit

**Gambar 4.3.4.1.** Cuplikan pertama Basis Data Biodata yang terenkripsi

tVsifyh6aSE9PkQKuxkFRw==	3AtjQStRNmCjYUEPgmyINw==	+QhzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
ZkwxbC9+eyFwMEsFTBckCSA==	JldjabRNng00AOYyFFEg==	+wh1PudvNXTNAHA1gyw1cXAwSwVMFwpI	<input type="checkbox"/> edit
x1xjRl2Z2i+rTSjN293LHawSwVMFwpI	+whzNPKsNXTXHC3o+y81d3AwSwVMFwpI	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
ZVkvfcV+bcfU0OA0VXBjLg==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
pU0veJ7RbDwMEsFTBckCSA==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
dUohbG1+bcrlP0UL2oERg==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
rEwubh8p+eyFAPkULPncERg==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
kkoqDOf6bCQ==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
sFssZFF6dyxwMEsFTBckCSA==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
XVktah9+ZSVwMEsFTBckCSA==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
XVYDD0j2bu=	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
LVgtb0BwbDtnPQKWBgFRw==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
CFg2DEV/d20=	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
QlkxaCv+cCvwMEsFTBckCSA==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
ZkwxbC9+eyFwMEsFTBckCSA==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
ThknYMPxdxE4SyJ+xHFrM1xZMxmPkghfUoIDB8+VzQ==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
O0of3p6dBwMEsFTBckCSA==	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit
xTAAoVvAio=	3AtjQStRNmCjYUEPgmyINw==	+whzNPKsNXTUDHM1uSw3d3AwSwVMFwpI	<input type="checkbox"/> edit

**Gambar 4.3.4.2.** Cuplikan kedua Basis Data Biodata yang terenkripsi

#### 4.4. Analisis hasil pengujian

Test Case	Waktu Eksekusi Algoritma (ms)	
	KMP	BM
Test-Case 1 (Real)	12.537	10.4464
Test-Case 2 (Altered-Low)	6.7386	7.2359
Test-Case 3 (Altered-Medium)	7.7463	7.6325
Test-Case 4 (Altered-High)	160.0199	158.5882

Untuk algoritma *Knuth-Morris-Pratt* (KMP), terdapat dua proses dalam perjalanan algoritmanya, yakni pembuatan table *prefix* (fungsi *boundary*) dan juga operasi *string matching* itu sendiri. Untuk pembuatan fungsi *boundary*, kita melakukan iterasi dari pola pattern sehingga kompleksitas waktunya adalah  $O(m)$  dengan  $m$  adalah panjang pola. Untuk operasi *matching*-nya, juga dilakukan secara linear terhadap panjang pola sehingga kompleksitas waktunya adalah  $O(n)$  dengan  $n$  adalah panjang dari teks. Karena tidak ada perulangan ulang yang signifikan dalam KMP, algoritma ini memiliki jaminan performa yang stabil yakni linear dalam panjang teks dan pola.

Untuk algoritma Boyer-Moore (BM) juga memiliki dua proses dalam penjalanan algoritmanya, yakni pembuatan tabel Last Occurrence dari tiap jenis karakter, dan juga operasi matching itu sendiri. Untuk pembuatan tabel Last Occurrence, kompleksitas waktunya adalah  $O(m+\sigma)$  dengan  $m$  adalah panjang pattern dan  $\sigma$  adalah jumlah karakter unik. Untuk operasi matchingnya sendiri, kompleksitas waktu rata-ratanya adalah  $O(n/m)$  dengan  $n$  adalah panjang teks dan  $m$  adalah panjang pattern. Dalam kasus rata-rata, BM sangat cepat dan sering kali mendekati  $O(n/m)$ . Namun, dalam kasus terburuk, seperti ketika pola dan teks terdiri dari karakter yang berulang secara tidak menguntungkan, kompleksitas bisa mencapai  $O(mn)$ .

Dapat dilihat pada tabel, bahwa waktu eksekusi dari algoritma KMP dan BM memiliki selisih waktu yang kecil. Hal ini dikarenakan kompleksitas waktu dari kedua algoritma tersebut yang terbilang mirip. Untuk kasus string matching pada pencocokan sidik jari dimana karakternya berupa ASCII, algoritma BM lebih bagus

karena algoritma BM dapat melewatkannya banyak karakter dengan heuristiknya sehingga sangat menguntungkan. Walaupun kekurangan dari BM adalah ketika pattern mendekati ukuran teks, hal ini tidak didapati pada pencocokan sidik jari karena pattern hanya berukuran 4 karakter, sedangkan teks berukuran 309 karakter.

Strategi pemilihan pattern dengan string biner juga terbukti dapat mendeteksi suatu sidik jari yang dilakukan operasi alter terhadapnya. Operasi alter ini dapat berupa rotasi pada bagian tertentu sidik jari, pem-*blur*-an bagian sidik jari, dan lain sebagainya. Strategi ini dapat dengan tepat menentukan sidik jari yang cocok ketika salah satu atau kedua dari pattern bukan berasal dari daerah yang mengalami peng-*alter*-an. Hal ini tidak kita temui pada kasus alter *blur* karena pada daerah blur, jumlah transisi pada daerah tersebut sedikit. Akan tetapi, hal ini menjadi resiko ketika menemui daerah sidik jari yang dilakukan alter rotasi, dan kebetulan daerah itu memiliki jumlah transisi yang banyak. Hal ini dalam beberapa kasus akan membuat pencocokan tidak berhasil.

## **BAB 5**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Pencocokan sidik jari merupakan salah satu dari aplikasi algoritma *String Matching* pada dunia nyata. Pencocokan sidik jari ini berguna untuk mengetahui informasi-informasi yang berkaitan dengan suatu sidik jari yang awalnya tidak diketahui milik siapa. Pencocokan string disini menggunakan string ASCII yang berasal dari hasil preprocess suatu gambar sidik jari. Preprocess ini akan mengubah nilai pixel tiap gambar yang awalnya berwarna, menjadi nilai binary (1/0), dan kemudian mengubahnya menjadi ASCII. Setelah melewati tahap preprocess, kemudian akan dilakukan string matching untuk menentukan sidik jari yang sesuai dengan input pengguna

Operasi *String Matching* ini dapat menggunakan dua algoritma string matching terkenal, yakni algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore (BM). Kedua algoritma ini menawarkan string matching dengan kompleksitas yang linear dengan kelebihan dan kekurangannya masing-masing. Pencocokan string ini kemudian akan menentukan exact match antara sidik jari input dengan sidik jari pada database. Jika string pattern ditemukan pada string ASCII dari sidik jari pada database, maka sidik jari tersebut merupakan sidik jari yang cocok dengan input pengguna sehingga kita dapat mengetahui biodata sidik jari yang sebelumnya telah diketahui pada database.

Data korup pada database juga dipertimbangkan pada projek ini, dimana suatu data dapat menjadi korup dengan adanya penggunaan bahasa-bahasa alay. Pada projek ini, data-data korup ini telah ditangani dengan penggunaan regex sehingga tetap dapat dilakukan pencocokan nama walaupun data nama pada database korup.

#### **5.2. Saran**

Secara umum, aplikasi yang dikembangkan oleh kelompok Selesaikan memiliki fungsionalitas yang cukup baik dengan menggunakan algoritma-algoritma, seperti *Knuth-Morris-Pratt* dan *Boyer Moore*. Selain itu, juga menggunakan beberapa kakas-kakas modern yang sangat membantu proses pengembangan yang telah dilakukan. Ditambah lagi, kelompok Selesaikan juga berhasil mengerjakan beberapa spesifikasi tambahan dalam penyelesaian tugas besar III ini dengan menambahkan fitur enkripsi dan dekripsi pada basis data

Biodata dengan menggunakan algoritma *blowfish*, serta membuat *video* penjelasan serta mempromosikan aplikasi yang telah dikembangkan oleh kelompok Selesaikan. Untuk menambah kecepatan pencarian pencocokan sidik jari yang secara repetitif pada *dataset* yang sama, kelompok Selesaikan mengantisipasi hal tersebut dengan melakukan *caching* sehingga pencarian berulang tidak akan memakan waktu yang lama, sehingga mempercepat dan memudahkan pengguna aplikasi.

Namun, tentu saja di balik fitur-fitur yang telah dicapai, terdapat beberapa evaluasi dan saran yang dapat ditambahkan untuk membuat kualitas aplikasi menjadi lebih baik daripada yang telah selesai dikembangkan. Saran kepada pengembang untuk memanfaatkan waktu yang ada dengan sebaik mungkin untuk mencari *bug* dan *edge case* sedini mungkin sehingga tidak menimbulkan *bug* pada *final testing* aplikasi sebelum aplikasi pada akhirnya dirilis menjadi sebuah aplikasi yang telah dianggap baik oleh pengembang.

Selain saran untuk cara kerja yang digunakan oleh pengembang dalam mengembangkan aplikasi ini, terdapat juga saran terhadap modularitas kode *frontend* yang dapat diperhatikan sehingga dapat meningkat *readability* serta memanfaatkan sifat-sifat dari OOP sebaik mungkin. Selain itu, terdapat juga saran untuk mengembangkan GUI untuk menjadi lebih interaktif dan kreatif dibandingkan desain yang monoton dan hanya memenuhi fungsi utama / *requirements* dari spesifikasi tugas yang telah diberikan.

### 5.3. Tanggapan dan Refleksi

Dalam tugas besar ini, seluruh anggota dari kelompok Selesaikan merasa bersyukur atas hasil atas usaha yang telah dikeluarkan untuk mengembangkan aplikasi yang disesuaikan dengan spesifikasi tugas (*requirements*) yang telah diberikan. Dari tugas besar ini, kami menemukan banyak tantangan dan rintangan, namun dapat kita hadapi dan tangani dengan baik. Meskipun satu sama lain dari kami terpisah oleh jarak, namun komunikasi anggota kelompok ini tidak pernah terputus sehingga dapat menyelesaikan tugas besar ini dengan lancar.

Secara ilmu, seluruh anggota kelompok mendapatkan pengetahuan dan wawasan yang luas serta dalam terkait mengimplementasi algoritma *pattern matching* berupa algoritma *Knuth-Morris-Pratt* (KMP), algoritma *Boyer Moore* (BM), dan *regular expression*. Telah dipelajari implementasi ketiga hal tersebut secara ril melalui tugas besar ini. Selain itu, juga didapatkan pengetahuan serta pengalaman dalam pengembangan sebuah aplikasi yang memiliki

GUI dengan menggunakan bahasa pemrograman C#, yang mana merupakan sebuah *experience* yang unik dan baru. Kemudian, juga digunakan *framework* seperti WPF dan *tools-tools* lain seperti Adminer yang merupakan sebuah kertas yang baru pertama kali untuk digunakan oleh beberapa anggota pada kelompok Selesaikan ini.

Dengan begitu, semoga ilmu pengetahuan dan pengalaman yang telah didapatkan oleh masing-masing anggota pada kelompok Selesaikan ini seyogianya dapat dipergunakan dengan baik di masa yang akan datang. Secara keseluruhan, tugas besar ini merupakan sebuah tugas yang cukup menantang dan berhasil memberikan sensasi yang unik dan seru terhadap seluruh anggota kelompok Selesaikan.

## **LAMPIRAN**

### **Pranala Repository**

[https://github.com/mybajwk/Tubes3\\_Selesaikan/commits/main/](https://github.com/mybajwk/Tubes3_Selesaikan/commits/main/)

### **Pranala Video**

<https://youtu.be/rO6qei9cfeE>

### **Dataset *Fingerprint* yang digunakan**

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

## **DAFTAR PUSTAKA**

Munir, Rinaldi. n.d. “Pencocokan string (String matching/pattern matching)” Informatika.

Diakses 24 April 2024.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf>

Munir, Rinaldi. n.d. “Pencocokan string dengan Regular Expression (Regex)” Informatika.

Diakses 24 April 2024.

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf>

GeeksforGeeks. n.d. “Introduction to Levenshtein Distance.” Diakses 24 April 2024.

<https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/>