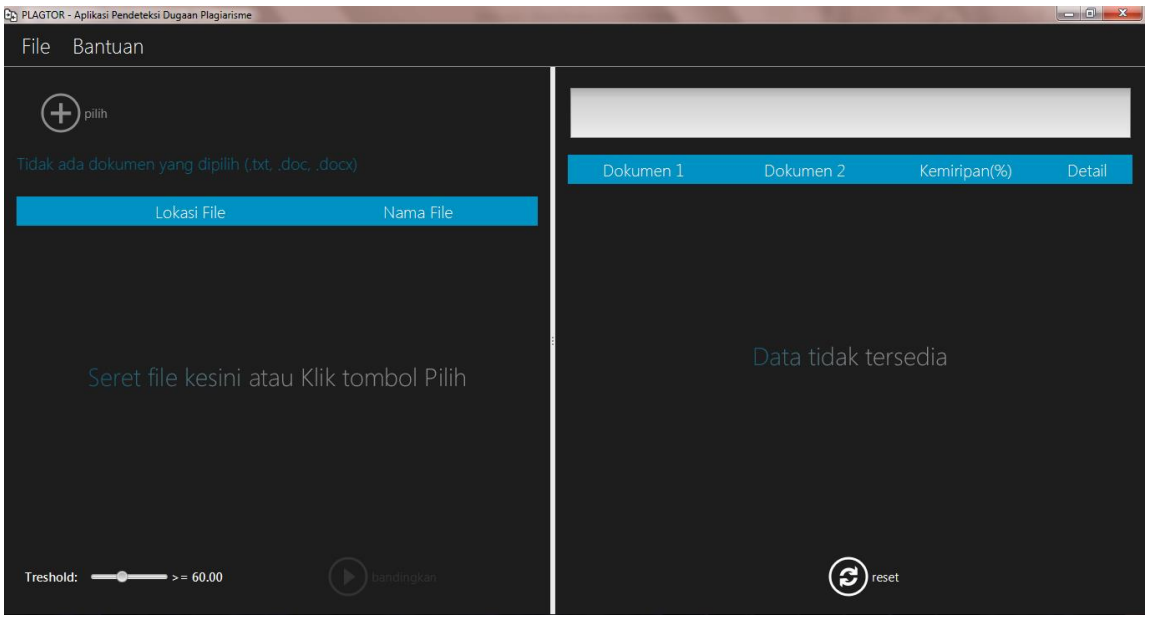
BAB IV

HASIL PENELITIAN DAN PEMBAHASAN

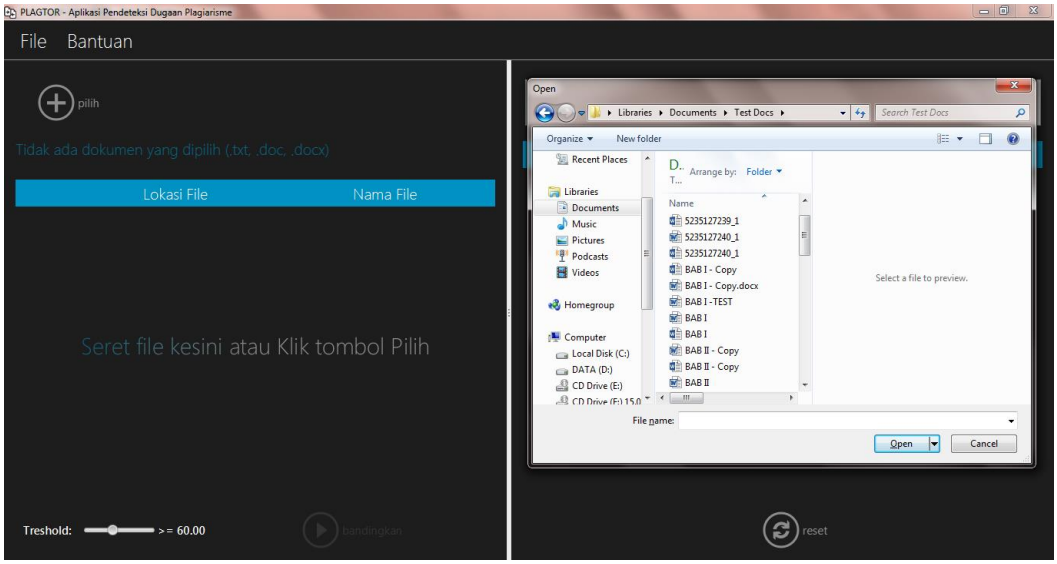
4.1. Hasil Penelitian

4.1.1. Implementasi *User Interface* Aplikasi

Berdasarkan rancangan *user interface* aplikasi pada bab sebelumnya, maka dihasilkan tampilan *user interface* seperti pada gambar 4.1. Pada halaman utama ini tersedia beberapa fitur, salah satunya *button* untuk meng-*upload* dokumen-dokumen yang hendak dibandingkan seperti pada gambar 4.2. Setelah itu, *user* juga dapat mengatur nilai *threshold* sebagai nilai ambang batas perbandingan kalimat yang akan diproses lebih lanjut.

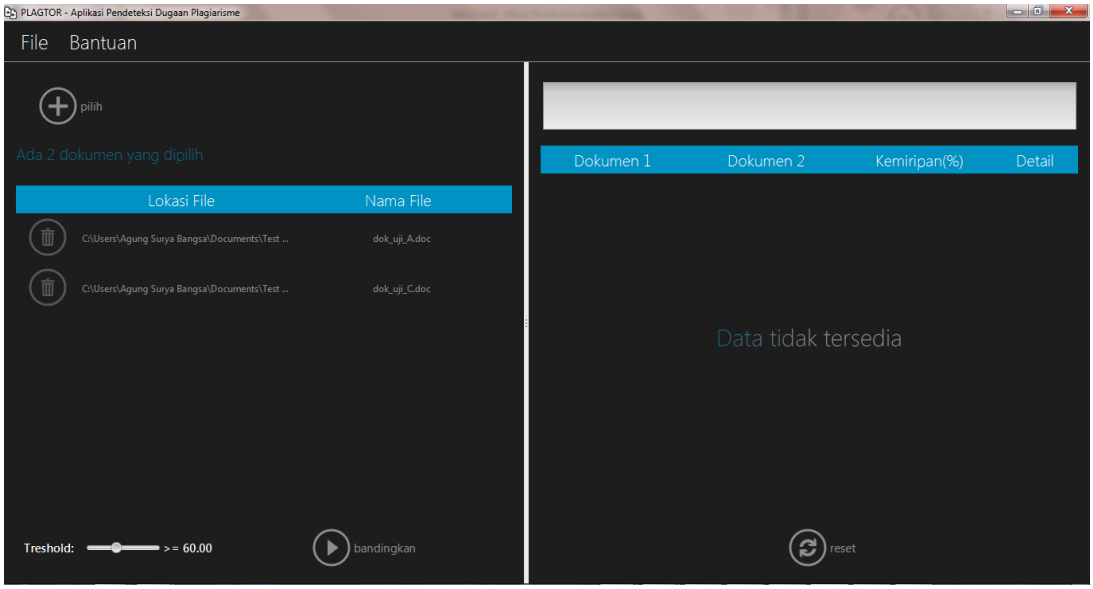


Gambar 4.1 *User interface* halaman utama



Gambar 4.2 *User interface* untuk *upload file*

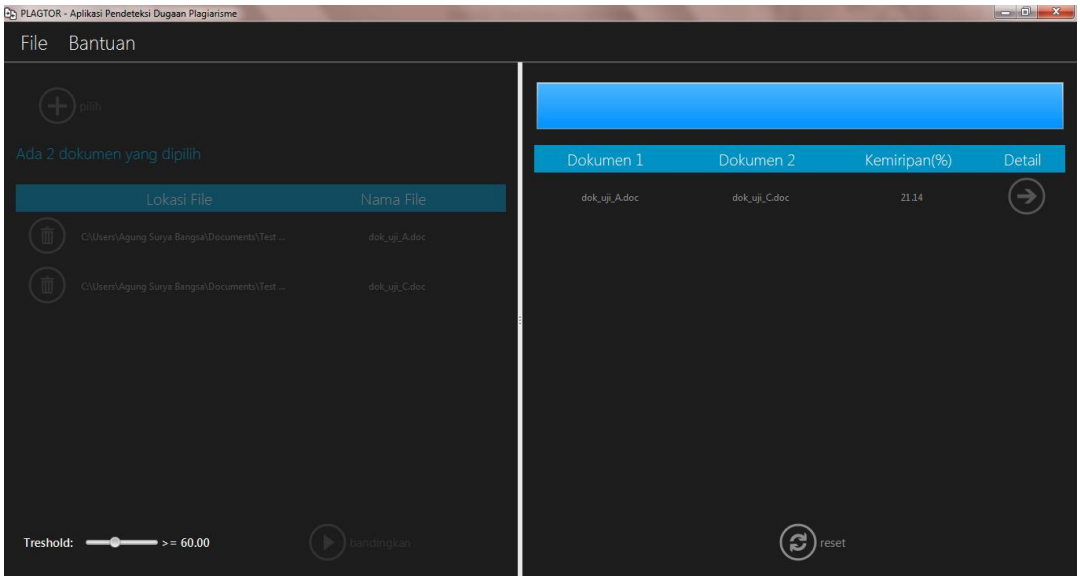
Agar proses perbandingan dapat dilakukan, *user* harus meng-*upload* minimal 2 buah *file* seperti yang ditunjukkan pada gambar 4.3.



Gambar 4.3 *User interface* setelah *upload file*

Jika jumlah *file* yang di-*upload* sudah memenuhi syarat minimum, maka langkah selanjutnya adalah melakukan proses penghitungan dugaan kemiripan

dokumen. *Output* yang disajikan adalah berupa persentase dugaan kemiripan (*similarity*) antar dokumen yang dibandingkan seperti yang ditunjukkan oleh gambar 4.4.



Gambar 4.4 *User interface* hasil akhir

4.1.2. Implementasi Sistem dan Algoritma

4.1.2.1. Tahap *Preprocessing*

Langkah yang dilakukan sebelum proses penghitungan adalah membaca *file input*. *File* berekstensi .txt, .doc, maupun .docx merupakan tipe *file* yang dapat diproses oleh aplikasi ini. *File-file* yang masuk dikonversi menjadi *temporary file* berekstensi .txt. Untuk *parsing* isi dokumen, aplikasi ini menggunakan *library* Aspose Word for Java dengan sedikit penyesuaian. Gambar 4.5. berikut adalah *source code* konversi dokumen yang masuk menjadi .txt.

```
public class KonversiDokumen {
    public static void main(String[] args) throws Exception {
    protected String keTxt(String SelectedFiles) throws Exception {
        try {
            Document doc = new Document(SelectedFiles);
            String isi = doc.getText();
            WriteFile(SelectedFiles, isi);
        } catch (IOException ioe) {
            System.out.println(ioe);
        }
        String TxtFile = SelectedFiles + ".txt";
        return TxtFile;
    }
    protected void WriteFile(String WrittenFile, String isi) {
        String fileName = WrittenFile + ".txt";
        try {
            FileWriter fileWriter = new FileWriter(fileName);
            try (BufferedWriter bufferedWriter = new BufferedWriter(fileWriter))
{
                bufferedWriter.write(isi);
            }
        } catch (IOException ex) {
            System.out.println("Error writing to file '" + fileName + "'");
        }
    }
}
```

Gambar 4.5. Konversi dokumen ke .txt.

Langkah selanjutanya adalah melakukan pembacaan *file* dengan *class* ReadFile. *Class* ini berfungsi membaca dan melakukan *parsing file* .txt hasil konversi proses sebelumnya. Gambar 4.6 berikut adalah *source code* dari class ReadFile.

```
public class ReadFile {
    public String[] OpenFile() throws IOException {
        FileReader fr = new FileReader(path);
        BufferedReader textReader = new BufferedReader(fr);
        int numberOfLines = readLines();
        String[] textData = new String[numberOfLines];
        for (int i = 0; i < numberOfLines; i++) {
            textData[i] = textReader.readLine();
        } textReader.close();
        return textData;
    }
    int readLines() throws IOException {
        FileReader file_to_read = new FileReader(path);
        BufferedReader bf = new BufferedReader(file_to_read);
        String aLine;
        int numberOfLines = 0;
        while ((aLine = bf.readLine()) != null) {
            numberOfLines++;
        } bf.close();
        return numberOfLines;
    }
}
```

Gambar 4.6. *Source code* dari *class* ReadFile

Setelah proses ReadFile, maka sistem menghitung jumlah token kalimat pada dokumen. Proses ini berguna untuk menentukan faktor pembagi pada proses akhir nanti.

```
public static int hitungKalimat(String[] aryLines, ArrayList<String>
JoinParagrafDoc, String[] KalimatDoc, String[] perKalimat) {
        String regex = "\\. ";
        for (int i = 0; i < aryLines.length; i++) {
              JoinParagrafDoc.add(aryLines[i]);
        }
        System.out.println("Dokumen Satu: " + JoinParagrafDoc);
        StringBuilder sb = new StringBuilder();
        for (String s : JoinParagrafDoc) {
               sb.append(s).append(" ");
         }
        String TextDoc1 = sb.toString();
        TextDoc1 = TextDoc1.replaceAll("\\s+", " ").trim();
        String[] perKalimat1 = TextDoc1.split(regex);
        KalimatDoc = (String[]) Metode.resizeArray(KalimatDoc,
perKalimat.length);
        for (int i = 0; i < perKalimat.length; i++) {
               KalimatDoc[i] = perKalimat[i];
        }
        return KalimatDoc.length;
}
```

Gambar 4.7. *Source code* dari *method* hitungKalimat()

Dari *source code* diatas, dapat dilihat bahwa dokumen dipecah menggunakan *delimiter* ". " sehingga menjadi token-token kalimat. Proses ini dinamakan dengan

tokenizing. Berikutnya, token-token kalimat itu akan mengalami *case folding* dan *sorting*. *Sorting* dilakukan dengan method urutkanKata(). Gambar 4.8 berikut adalah *source code* untuk method urutkanKata().

```
public class treeSet {


    protected static String[] urutkanKata(String[] text) {
        SortedSet<String> mySortedSet = new TreeSet<String>();


        for (int i = 0; i < text.length; i++) {
            mySortedSet.add(text[i]);
        }


        String[] teks = mySortedSet.toArray(new String[mySortedSet.size()]);
        //String[] result = myTreeSet.toArray(new String[myTreeSet.size()]);
        int idx = 0;
        for (String it : mySortedSet) {
            teks[idx] = it.toString();
            idx++;
        }
        return teks;
    }
}
```

Gambar 4.8. *Source code* dari *method* urutkanKata()

Proses *sorting* ini menggunakan fungsi *SortedSet* yang ada pada bahasa pemrograman Java. *Interface SortedSet* ini berfungsi menjaga agar elemen dalam

kondisi terurut. Pengurutan elemen ini sangat diperlukan untuk mengoptimalkan kerja algoritma Levenshtein *distance* dalam mendeteksi kemiripan teks.

4.1.2.2. Tahap Penghitungan Dugaan Kemiripan Isi Teks

Tahap ini adalah tahap penghitungan dugaan kemiripan dokumen dengan algoritma Levenshtein *distance*. Untuk mendapatkannya, langkah-langkah yang dilakukan adalah:

1) Elemen matriks[i][0], diisi dengan elemen i.
2) Elemen matriks[0][j], diisi dengan elemen j.
3) Elemen matriks[i][j], diisi dengan urutan langkah sebagai berikut:
    1) Jika karakter ke-i pada string ke-1 memiliki kesamaan dengan karakter ke-j pada string ke-2, maka elemen matriks[i][j] diisi dengan elemen matriks[i-1][j-1].
    2) Jika karakter ke-i pada string ke-1 berbeda dengan karakter ke-j pada string ke-2, maka elemen matriks[i][j] diisi dengan nilai terkecil dari perbandingan tiga elemen matriks[i-1][j], matriks[i][j-1], dan matriks[i-1][j-1].
    3) Ulangi langkah a dan b hingga perbandingan selesai untuk semua karakter.

Gambar 4.9 berikut merupakan *source code* untuk *setup* matriks pada algoritma Levenshtein *distance* menggunakan *method* setupMatrix().

```
private void setupMatrix() {
    matrix = new int[compOne.length() + 1][compTwo.length() + 1];
    for (int i = 0; i <= compOne.length(); i++) { matrix[i][0] = i; }
    for (int j = 0; j <= compTwo.length(); j++) { matrix[0][j] = j; }
    for (int i = 1; i < matrix.length; i++) {
        for (int j = 1; j < matrix[i].length; j++) {
            if (compOne.charAt(i - 1) == compTwo.charAt(j - 1)) {
                matrix[i][j] = matrix[i - 1][j - 1];
            } else {
                int minimum = Integer.MAX_VALUE;
                if ((matrix[i - 1][j]) + 1 < minimum) {
                    minimum = (matrix[i - 1][j]) + 1;
                }
                if ((matrix[i][j - 1]) + 1 < minimum) {
                    minimum = (matrix[i][j - 1]) + 1;
                }
                if ((matrix[i - 1][j - 1]) + 1 < minimum) {
                    minimum = (matrix[i - 1][j - 1]) + 1;
                }
                matrix[i][j] = minimum;
            }
        }
    } calculated = true;
}
```

Gambar 4.9. *Source code* untuk *setup* matriks

Langkah berikutnya adalah penghitungan *similarity* dengan *method* getSimilarity().

```
public int getSimilarity() {
    if (!calculated) {
        setupMatrix();
    }
    return matrix[compOne.length()][compTwo.length()];
}
```

Gambar 4.10. *Source code* untuk *method* getSimilarity()

Dalam menghitung nilai *similarity*, *method* getSimilarity() memanggil *method* setupMatrix(), kemudian mengembalikan nilai Levenshtein *distance* yang disimpan dalam variabel matriks. Kemudian, nilai yang dikembalikan ini akan digunakan untuk penghitungan *plagiarized value* pada fungsi utama penghitungan dugaan kemiripan dokumen. Penghitungan dilakukan dalam *looping* yang jumlahnya sebanyak perkalian total kalimat yang terdapat pada dokumen yang sedang dibandingkan. *Plagiarized value* yang sama dengan atau melebihi *threshold* akan disimpan dalam sebuah variabel secara akumulatif. Terakhir, nilai akumulatif tersebut dibagi dengan nilai maksimum antara *plagiarized value* yang lolos *threshold* atau banyaknya total kalimat maksimum diantara dokumen-dokumen atau yang dibandingkan. Rumusnya adalah sebagai berikut:

$$Total\ Plagiarized\ Value = \left\{\frac{Akumulasi\ PV}{Max(Max(CS, ST),\ Lolos)}\right\}$$

Keterangan:

Lolos = Banyaknya pasangan kalimat yang lolos *threshold*

Gambar 4.12 adalah *source code* untuk menghitung *plagiarized value* dokumen.

```
int lolos = 0;
for (int x = 0; x < KalimatDoc1.length; x++) {
            for (int w = 0; w < KalimatDoc2.length; w++) {
               String[] ax = KalimatDoc1[x].toLowerCase().split("\\ ");
               String[] ad = treeSet.urutkanKata(ax);
               String[] bx = KalimatDoc2[w].toLowerCase().split("\\ ");
               String[] bd = treeSet.urutkanKata(bx);
               if (KalimatDoc1[x] != null || KalimatDoc2[w] != null) {
                     Levenshtein l = new
Levenshtein(Arrays.toString(ad).replaceAll("\\W", ""),
Arrays.toString(bd).replaceAll("\\W", ""));
                     double st = Arrays.toString(ad).replaceAll("\\W", "").length();
                     double cs = Arrays.toString(bd).replaceAll("\\W", "").length();
                     double pv = (1 - (l.getSimilarity() / Math.max(st, cs)));
                     if (pv >= threshold) {
                        lolos++;
                        TotalKesamaan = (TotalKesamaan + pv);
                     }
               }
         }
}
int faktorPembagi = Math.max(lolos, Math.max((KalimatDoc1.length),
(KalimatDoc2.length)));
plagiarize_value = (TotalKesamaan / faktorPembagi) * 100;
System.out.println("Total Plagiarized value: " + plagiarize_value);
```

Gambar 4.11. *Source code* untuk penghitugan *plagiarized value* dokumen

Dalam proses penghitungan diatas, semua fungsi yang ada dalam tahap *preprocessing* sudah dilakukan, dimulai dari *read file, tokenizing, case folding*¸ dan *sorting*. Fungsi diatas akan memberikan *output* akhir aplikasi berupa persentase *plagiarized value* dokumen yang dibandingkan.

4.1.3. Uji Coba

4.1.3.1. Skenario Pengujian

Uji coba terhadap aplikasi ini dilakukan dengan menggunakan dua jenis dokumen, yaitu dokumen yang struktur kalimat didalam paragrafnya diubah dan dokumen yang struktur kata didalam kalimatnya diubah.

1) Data Uji 1 (Pengubahan struktur kalimat didalam paragraf)

   Pengujian ini dilakukan dengan membandingkan dua dokumen yang sama tetapi telah diubah letak struktur kalimatnya. Misalnya, dengan mengubah paragraf induktif menjadi deduktif atau sebaliknya. Berikut contoh pengubahan struktur kalimat didalam paragraf:

   1) Paragraf A:

Banyak penjual kerudung di jalan keluar pabrik. Di sisi-sisi penjual itu juga nampak penjual kurma dan buah lainnya. Penjual makanan juga memenuhi trotoar jalan, yang sebagian besar adalah penjual dadakan. Mulai dari menu takjil hingga makanan berbuka puasa tersedia disini. Belum lagi penjual pakaian yang memanjang hingga ujung jalan. Kerumunan pedagang tersebut makin membuat lalu lintas padat. Terlebih ketika para pekerja pulang dari pabrik, tetapi hal itu dapat dimaklumi. Memang, setiap menjelang Hari Raya Idul Fitri banyak orang menjadi pedagang karena keuntungannya yang besar

2) Paragraf B (Paragraf A yang ditukar letak kalimatnya):

> Menjelang Hari Raya Idul Fitri banyak orang menjadi pedagang karena keuntungannya yang besar. Misalnya saja di lingkungan pabrik. Banyak ditemukan penjual kerudung di jalan keluar pabrik. Penjual kurma dan buah juga nampak di sisi-sisi penjual itu. Mulai dari menu takjil hingga makanan berbuka puasa tersedia disini. Penjual makanan tersebut memenuhi trotoar jalan, yang sebagian besar penjual dadakan. Belum lagi penjual pakaian yang memanjang hingga ujung jalan. Kerumunan pedagang tersebut makin membuat lalu lintas padat. Terlebih ketika para pekerja pulang dari pabrik, tetapi hal itu dapat dimaklumi.

2) Data Uji 2 (Pengubahan struktur kata didalam kalimat)

Pengujian ini dilakukan dengan membandingkan dua dokumen yang struktur kata didalam kalimatnya telah diubah, yaitu dengan mengubah kalimat pasif menjadi aktif atau sebaliknya, serta dengan penukaran posisi kata dan perubahan kata berimbuhan. Kondisi dokumen pembanding pada data uji coba ini mirip dengan tindakan plagiarisme mosaik, dimana plagiarisme tidak dilakukan kata demi kata, melainkan diselang-seling atau disisip-sisipkan sehingga memberikan kesan bahwa dokumen yang dihasilkan adalah dokumen asli. Berikut ini merupakan contoh pengubahan struktur kata didalam kalimat:

1) Adik membeli mainan di toko tadi pagi.
2) Tadi pagi adik membeli mainan di toko.
3) Mainan dibeli adik di toko tadi pagi.
4) Tadi pagi mainan dibeli adik di toko.

Contoh Data Uji 2 dalam paragraf:

1) Paragraf A

> Beberapa kelebihan dimiliki kelengkeng Pingpong jika dibandingkan dengan kelengkeng lain sehingga sangat cocok untuk dibudidayakan. Ukuran buah sebesar bola pingpong dimiliki oleh kelengkeng pingpong. Kelengkeng Pingpong dapat berbuah dengan cepat. Pada usia 1,5 tahun sudah bisa mulai berbuah. Bila buah-buahan lain rata-rata berbuah setahun sekali, kelengkeng pingpong berbuah lebih dari sekali dalam setahun. Karena buah bisa tumbuh dari ranting-ranting secara bergantian, boleh dikatakan Kelengkeng Pingpong berbuah dengan tidak mengenal musim. Kelengkeng Pingpong bisa ditanam pada media pot, bagi masyarakat perkotaan yang memiliki lahan terbatas. Kelengkeng Pingpong menjadi potensi agrobisnis yang cukup menggiurkan karena kelebihan-kelebihan tersebut.

2) Paragraf B

> Beberapa kelebihan dimiliki kelengkeng Pingpong jika dibandingkan dengan kelengkeng lain sehingga sangat cocok untuk dibudidayakan. Kelengkeng Pingpong memiliki ukuran buah yang besar sebesar bola pingpong. Selain itu bisa tumbuh di dataran rendah. Kelengkeng Pingpong dapat berbuah dengan cepat. Pada usia 1,5 tahun sudah bisa mulai berbuah. Kelengkeng Pingpong dapat berbuah lebih dari satu kali dalam satu tahun. Sedangkan buah-buahan lain rata-rata berbuah satu tahun sekali. Karena buah bisa tumbuh dari ranting-ranting secara bergantian, boleh dikatakan Kelengkeng Pingpong berbuah dengan tidak mengenal musim. Kelengkeng Pingpong bisa ditanam pada lahan sempit bahkan media pot, bagi masyarakat perkotaan yang memiliki lahan terbatas. Kelengkeng Pingpong menjadi potensi agrobisnis yang cukup menggiurkan karena kelebihan-kelebihan tersebut.

4.1.3.2. Hasil Pengujian

Uji coba dilakukan dengan menguji data menggunakan sistem aplikasi yang telah dijabarkan. Nilai *threshold* yang digunakan adalah >= 60.

1) Hasil Uji Coba Data Uji 1

Berdasarkan skenario uji coba data uji 1, maka proses penghitungan paragraf A dengan paragraf B adalah sebagai berikut:

A1: banyakdijalankeluarkerudungpabrikpenjual

B3: banyakdiditemukanjalankeluarkerudungpabrikpenjual

Levenshtein *distance* = 9

Plagiarized value = 81,63%

A2: buahdandiitujugakurmalainnyanampakpenjualsisisisi

B4: buahdandiitujugakurmanampakpenjualsisisisi

Levenshtein *distance* = 7

Plagiarized value = 85,71%

A3: adalahbesardadakanjalanjugamakananmemenuhipenjualsebagiantrotoaryang

B6: besardadakanjalanmakananmemenuhipenjualsebagiantersebuttrotoaryang

Levenshtein *distance* = 18

Plagiarized value = 73,52%

A4: berbukadaridisinihinggamakananmenumulaipuasatakjiltersedia

B5: berbukadaridisinihinggamakananmenumulaipuasatakjiltersedia

Levenshtein *distance* = 0

Plagiarized value = 100%

A5: belumhinggajalanlagimemanjangpakaianpenjualujungyang

B7: belumhinggajalanlagimemanjangpakaianpenjualujungyang

Levenshtein *distance* = 0

Plagiarized value = 100%

A6: kerumunanlalulintasmakinmembuatpadatpedagangtersebut

B9: kerumunanlalulintasmakinmembuatpadatpedagangtersebut

Levenshtein *distance* = 0

Plagiarized value = 100%

A7: dapatdaridimaklumihalituketikapabrikparapekerjapulangterlebihtetapi

B9: dapatdaridimaklumihalituketikapabrikparapekerjapulangterlebihtetapi

Levenshtein *distance* = 0

Plagiarized value = 100%

A8: banyakbesarfitrihariidulkarenakeuntungannyamemangmenjadimenjelangorangpedagangrayasetiapyang

B1: banyakbesarfitrihariidulkarenakeuntungannyamenjadimenjelangorangpedagangrayayang

Levenshtein *distance* = 12

Plagiarized value = 86,95%

Dari perbandingan kalimat diatas dapat dihitung nilai akhir dugaan kemiripan dokumen dengan menjumlahkan semua *plagiarized value* dan

membaginya dengan nilai maksimum antara *plagiarized value* yang lolos *threshold* dan jumlah kalimat terbanyak dari paragraf A dan paragraf B.

Jumlah kalimat paragraf A = 8

Jumlah kalimat paragraf B = 9

Lolos = 8, Max(A, B) = 9.

Nilai akhir = $\left\{\frac{Akumulasi\ PV}{Max(Max(CS,ST),Lolos)}\right\}$

= $\left\{\frac{81{,}63 + 85{,}71 + 73{,}52 + 100 + 100 + 100 + 100 + 86{,}95}{Max(9{,}8)}\right\}$

= $\left\{\frac{727.81}{9}\right\}$

= 80,87

Dengan demikian, dari hasil nilai akhir diatas dapat disimpulkan bahwa dugaan kemiripan dokumen yang dibandingkan adalah sebesar 80,87%.

2) Hasil Uji Coba Data Uji 2

Berdasarkan skenario uji coba data uji 2, maka proses penghitungan paragraf A dengan paragraf B adalah sebagai berikut:

A1: beberapacocokdengandibandingkandibudidayakandimilikijikakelebihankelengkenglainpingpongsangatsehinggauntuk

B1: beberapacocokdengandibandingkandibudidayakandimilikijikakelebihankelengkenglainpingpongsangatsehinggauntuk

Levenshtein *distance* = 0

Plagiarized value = 100%

A3: berbuahcepatdapatdengankelengkengpingpong

B4: berbuahcepatdapatdengankelengkengpingpong

Levenshtein *distance* = 0

Plagiarized value = 100%

A4: 15berbuahbisamulaipadasudahtahunusia

B5: 15berbuahbisamulaipadasudahtahunusia

Levenshtein *distance* = 0

Plagiarized value = 100%

A6: berbuahbergantianbisabolehbuahdaridengandikatakankarenakelengkengmengenalmusimpingpongrantingrantingsecaratidaktumbuh

B8: berbuahbergantianbisabolehbuahdaridengandikatakankarenakelengkengmengenalmusimpingpongrantingrantingsecaratidaktumbuh

Levenshtein *distance* = 0

Plagiarized value = 100%

A7:
bagibisaditanamkelengkenglahanmasyarakatmediamemilikipadaperkotaanpingpongpotterbatasyang

B9:
bagibahkanbisaditanamkelengkenglahanmasyarakatmediamemilikipadaperkotaanpingpongpotsempitterbatasyang

Levenshtein *distance* = 12

Plagiarized value = 88,12%

A8:
agrobisniscukupkarenakelebihankelebihankelengkengmenggiurkanmenjadipingpongpotensitersebutyang

B10:
agrobisniscukupkarenakelebihankelebihankelengkengmenggiurkanmenjadipingpongpotensitersebutyang

Levenshtein *distance* = 0

Plagiarized value = 100%

Lolos = 6, Max(A, B) = 10.

Nilai akhir = $\left\{\frac{Akumulasi\ PV}{Max(Max(CS,ST),\ Lolos)}\right\}$

= $\left\{\frac{100 + 100 + 100 + 100 + 88{,}12 + 100}{Max(10,\ 6)}\right\}$

= $\left\{\frac{588{,}11}{10}\right\}$

= 58,81

Dengan demikian, dari hasil uji coba diatas, dipeoleh nilai persentase dugaan kemiripan dokumen yang dibandingkan sebesar 58,81%.

3) *Running Time* Aplikasi

Tabel berikut menunjukkan hasil pengujian terhadap dokumen uji yang terdapat pada subbab 3.4.2.3 menggunakan aplikasi.

Tabel 4.1 Tabel hasil pengujian dokumen uji

No	Dokumen A	Dokumen B	Nilai Akhir (%)
1.	dok_uji_B	dok_uji_D	100
2.	dok_uji_A	dok_uji_B	78,81
3.	dok_uji_A	dok_uji_D	78,81
4.	dok_uji_A	dok_uji_C	21,14

Running time proses penghitungan persentase kemiripan dokumen yang dibandingkan dapat dilihat pada tabel berikut:

Tabel 4.2 Tabel perhitungan *running time*

No	Total Kata Dokumen Teks A	Total Kata Dokumen Teks B	Persentase Kemiripan	Running Time (detik)
1.	84	86	80,87	1,071
2.	95	110	58,81	1,037
3.	276	277	89,56	0,57
4.	371	368	93,47	0,442
5.	543	540	92,48	0,104
6.	330	505	5,3	0,597
7.	341	715	24,12	0,717
8.	1220	847	76,48	0,182
9.	2050	1715	14,81	0,12
10.	1519	1657	0	0,382

4) Hasil Uji Coba Perbandingan Manual dengan Aplikasi

Untuk mengetahui tingkat akurasi *output* aplikasi, maka hasil uji coba perbandingan menggunakan aplikasi dibandingkan dengan hasil uji coba secara manual. Total dokumen yang dibandingkan berjumlah 30 dokumen,

6 diantaranya diambil sebagai *sample*. Data yang ditampilkan adalah data perbandingan 6 dokumen *sample* dengan dokumen lainnya. Nilai *threshold* yang diterapkan aplikasi pada uji coba ini adalah 60. Isi teks dari dokumen-dokumen yang digunakan dapat dilihat pada lampiran.

Tabel 4.3 Tabel hasil pengujian dokumen tugas siswa secara manual

MANUAL	HASIL PERBANDINGAN					
	dokuji1	**dokuji2**	**dokuji9**	**dokuji17**	**dokuji21**	**dokuji24**
dokuji1	-	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji2	Tidak	-	Tidak	Tidak	Tidak	Tidak
dokuji3	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji4	Mirip	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji5	Mirip	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji6	Mirip	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji7	Mirip	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji8	Tidak	Mirip	Tidak	Tidak	Tidak	Tidak
dokuji9	Tidak	Tidak	-	Mirip	Tidak	Tidak
dokuji10	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji11	Tidak	Mirip	Tidak	Tidak	Tidak	Tidak
dokuji12	Tidak	Mirip	Tidak	Tidak	Tidak	Tidak
dokuji13	Tidak	Mirip	Tidak	Tidak	Tidak	Tidak
dokuji14	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji15	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji16	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji17	Tidak	Tidak	Mirip	-	Tidak	Tidak
dokuji18	Tidak	Tidak	Mirip	Mirip	Tidak	Tidak
dokuji19	Tidak	Tidak	Mirip	Mirip	Tidak	Tidak
dokuji20	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji21	Tidak	Tidak	Tidak	Tidak	-	Tidak
dokuji22	Tidak	Tidak	Mirip	Mirip	Tidak	Tidak
dokuji23	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji24	Tidak	Tidak	Tidak	Tidak	Tidak	-
dokuji25	Tidak	Tidak	Tidak	Tidak	Mirip	Tidak
dokuji26	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji27	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji28	Tidak	Tidak	Mirip	Mirip	Tidak	Tidak
dokuji29	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak
dokuji30	Tidak	Tidak	Tidak	Tidak	Tidak	Tidak

Tabel 4.4 Tabel hasil pengujian dokumen tugas siswa dengan aplikasi

APLIKASI	HASIL PERBANDINGAN (%)					
	dokuji1	dokuji2	dokuji9	dokuji17	dokuji21	dokuji24
dokuji1	-	5.3	0.86	24.12	1.44	2.31
dokuji2	5.3	-	0	2.86	0	0
dokuji3	0	0	0	0	0	0
dokuji4	95.63	5.3	0.86	24.12	1.44	2.31
dokuji5	95.82	5.3	0.86	24.12	1.44	2.31
dokuji6	95.76	5.3	0.86	24.12	1.44	2.31
dokuji7	95.78	5.3	0.86	24.12	1.44	2.31
dokuji8	5.3	99.61	0	2.86	0	0
dokuji9	0.86	0	-	57.8	46.16	3.13
dokuji10	5.3	18.68	24.78	36.04	9.5	9.39
dokuji11	5.3	83.16	0	2.86	0	0
dokuji12	5.3	91.7	0	2.86	0	0
dokuji13	5.3	83.21	0	2.86	0	0
dokuji14	2.15	0	26.47	35.57	28.63	7.55
dokuji15	0	0	0	0	0	0
dokuji16	0	0	0	0	0	0
dokuji17	24.12	2.86	57.8	-	29.82	4.43
dokuji18	0.9	0	92.40	60.47	43.42	3.26
dokuji19	1.47	0	57.37	82.19	34.92	5.34
dokuji20	0	0	2.18	3.09	1.42	0
dokuji21	1.44	0	46.16	29.82	-	1.70
dokuji22	0.91	0	94.84	63.34	48.76	3.31
dokuji23	0	0	31.96	43.34	21.78	14.81
dokuji24	2.31	0	3.13	4.43	1.70	-
dokuji25	0	0	36.07	18.12	83.51	5.01
dokuji26	0	0	0	0	0	0
dokuji27	2.1	0	0.83	0	1.39	20.62
dokuji28	1.54	0	57.97	82.73	35.12	7.58
dokuji29	0	0	0	0	0	0
dokuji30	0	0	0	0	0	0

4.2. Pembahasan

Dari hasil percobaan yang dilakukan terhadap Data Uji 1, dapat disimpulkan bahwa aplikasi masih dapat mendeteksi kemiripan diantara kedua dokumen. Hal ini terbukti dengan tingginya persentase yang dihasilkan, yaitu 80,87%.

Sedangkan dari hasil percobaan yang dilakukan terhadap Data Uji 2 dimana dokumen pembandingnya memiliki struktur kata yang berbeda di dalam kalimatnya dibandingkan dengan dokumen aslinya, aplikasi tidak bisa mendeteksi dengan baik kemiripan dokumen dengan pola kalimat yang diubah-ubah letak susunan katanya. Hal ini terbukti dengan persentase kemiripan dokumen yang rendah, yaitu 58,81%.

Hasil uji coba perbandingan dokumen secara manual dengan perbandingan dokumen menggunakan aplikasi menunjukkan bahwa hasil perbandingan dokumen yang memiliki persentase diatas 57% pada perbandingan menggunakan aplikasi bernilai ‘mirip’ pada hasil perbandingan dokumen secara manual.