BAB 5 IMPLEMENTASI

5.1 Lingkungan Implementasi Sistem

Lingkungan implementasi akan dijelaskan mengenai implementasi algoritme menjadi suatu aplikasi atau *software* berbasis website. Implementasi yang dilakukan tentunya membutuhkan perangkat keras atau *hardware* dan perangkat lunak *software* yang sudang dirancang, serta mengenai lingkungan implementasi sistem yang akan digunakan oleh penulis dalam penelitian.

5.1.1 Lingkungan Perangkat Keras

Perangkat keras atau *hardware* yang digunakan dalam proses implementasi sistem. Penulis menggunakan perangkat keras yang dapat dilihat pada Tabel 5.1 berikut :

Tabel 5.1 Lingkungan Perangkat Keras

No	Keterangan	Detil
1	*Processor*	2.00 GHz Intel Core i3 - 6006U
2	*Memory* (RAM)	4 GB DDR3L 1600 MHz
3	L2 Cache	2 MB

5.1.2 Lingkungan Perangkat Lunak

Dalam melakukan penelitian ini, penulis menggunakan perangkat lunak atau *software* adalah seperti yang dapat dilihat pada Tabel 5.2 sebagai berikut:

Tabel 5.2 Lingkungan Perangkat Keras

No	Keterangan	Detil
1	Sistem Operasi	Windows 11 Pro 64-Bit
2	Bahasa Pemrograman	Python
3	Alat Pemrograman	Python 3.5.1 Shell
4	Dokumentasi	Microsoft Word Office 2013

5.2 Implementasi Data

Implementasi data yang dibuat ini berdasarkan dari perancangan data yang telah dibuat dan dijabarkan sebelumnya dalam subbab 4.1. pada implementasi data ini, digunakan sebnayak 10 dokumen dengan judul yang dapat dilihat pada lampiran

1. Sedangkan untuk data *stopword* yang digunakan pada proses filtering disimpan dalam file text dengan nama *stopword*.*txt.*

5.3Implementasi Algoritme

Pada implementasi algoritme ini akan membahas mengenai implementasi algoritme yang telah dilakukan. Dalam pengerjaan implementasi algoritme ini, penulis merujuk pada perancangan algoritme yang telah dijabarkan serta telah dibuat sebelumnya pada subbab 4.2

5.3.1 Implementasi *preprocessing*

Pada kode program yang tertera pada Tabel 5.1, baris 1 digunakan untuk memisahkan *string* pada kalimat yang didasarkan pada tanda baca titik (.) pada dokumen teks menggunakan *regex.*

Kode Program 5. 1 *Parsing*

No	Kode
1	`regex_kalimat = r"[A-Z][\w\s,!?\-\\\/()\&]+"`
2	`parsing = re.findall(regex_kalimat, beritaku)`
3	`for berita in parsing:`
4	`  print(berita)`

Pada Kode program 5.2 terdapat proses tokenisasi serta *filtering* pada kode baris ke-1 digunakan *regex* untuk mengambil karakter alphabet pada suatu dokumen teks. Proses tokenisasi dilakukan pada kode baris ke-4 yakni dengan mencocokkan *regex* dengan kalimat dari hasil *stemming*. Proses *filtering* ada pada baris ke-6 hingga ke-9 dengan menggunakan data hasil proses tokenisasi yang disimpan pada *list* kata_kalimat yang lalu dicocokkan dengan setiap *string* kata yang ada pada *stopword* dan yang tidak masuk dalam *list* token agar tidak terjadi duplikasi.

Kode Program 5. 2 Tokenisasi dan *Filtering*

No	Kode
1	`regex_tokenisasi = r"[a-z]+"`
2	`token = []`
3	`for kalimat in stemming:`
4	`  kata_kalimat =`
5	`re.findall(regex_tokenisasi, kalimat)`
6	`  for kata in kata_kalimat:`
7	`  if (kata not in token) and (kata not in`
8	`list_stopword):`
9	`  token.append(kata)`
10	`for item in token:`
11	`  print(item)`

Untuk Implementasi dari proses *stemming* Kode program 5.3 dengan menggunakan *library* atau modul yang telah tersedia pada *python* yaitu *library* Sastrawi. Pada kode program baris ke-3 merupakan proses *stemming* dari kalimat yang telah dilakukan proses parsing sebelumnya. Pada tahap ini juga terjadi proses *case folding* yang mengubah huruf kapital menjadi *lower case* atau huruf kecil. Sedangkan pada kode program baris ke-5 hingga ke-6 merupakan baris untuk mencetak hasil dari proses *stemming*.

Kode Program 5. 3 *Stemming*

No	Kode
1	`factory = StemmerFactory()`
2	`Stemmer = factory.create_stemmer()`
3	`stemming = [Stemmer.stem(kalimat)for kalimat in`
4	`parsing]`
5	`for berita in stemming:`
6	`  print (berita)`

Pada Kode program 5.3 baris ke-5 hingga baris ke-7 digunakan untuk menghitung kemunculan kata pada suatu dokumen TF yang tersimpan pada *list* token sebelumnya. pada kode baris ke-16 hasil TF dimasukkan dalam *list* TF. Sedangkan pada kode program baris ke-9 hingga pada baris ke-15 digunakan untuk menghitung kemunculan kata pada beberapa kalimat dengan menghitung setiap kalimat yang ada pada data sebelumnya yaitu *stemming*.

Kode Program 5. 4 Nilai *Raw* TF dan DF

No	Kode
1	`tf = []`
2	`list_df = []`
3	`ind = []`
4	`i = 0`
5	`for kata in token:`
6	`    temp = []`
7	`    temp.append(kata)`
8	`    df = 0`
9	`    for kalimat in stemming:`
10	`        if kata in kalimat:`
11	`            temp.append(1)`
12	`            df += 1`
13	`        else:`
14	`            temp.append(0)`
15	`    temp.append(df)`
16	`    tf.append(temp)`

```
17          list_df.append(df)
18          ind.append(i)
19          i += 1
20      for item in tf:
21        print (item)
```

Pada baris Kode program 5.3 ke-4 digunakan untuk mencari nilai IDF dan pada baris ke-7 hingga ke-9 digunakan untuk mencari nilai log tf berdasar nilai *index raw* TF sebelumnya.

Kode Program 5. 5 Nilai Log TF dan IDF

No	Kode
1	`   for item in tf:`
2	`       for index in range(1, len(item)):`
3	`           if index == len(item)-1:`
4	`               item[index]=`
5	`math.log10(len(stemming)/item[index])`
6	`           else:`
7	`               if item[index] > 0:`
8	`                 item[index]      =      1      +`
9	`math.log10(item[index])`

Pada Kode program 5.6 baris ke-5 sampai dengan baris ke-9 merupakan perhitungan TF-IDF yang mana nilai *log* TF dikalikan dengan DF yang disimpan pada *list tfIdf* yang dideklarasikan pada baris ke-1.

Kode Program 5. 6 Nilai Log TF.IDF

No	Kode
1	`   tfIdf = []`
2	`   for item in tf:`
3	`       temp = []`
4	`       temp.append(item[0])`
5	`       for index in range(1, len(item)):`
6	`           if index < len(item)-1:`
7	`               tf_idf                          =`
8	`item[index]*item[len(item)-1]`
9	`               temp.append(tf_idf)`
10	`    tfIdf.append(temp)`

5.3.2 Implementasi Perhitungan LSA

Pada Kode program 5.7 di baris ke-2 hingga baris ke-7 membentuk sebuah matriks A yang disimpan pada *list* A dari nilai hasil proses TF-IDF. Serta kode program

pada baris ke-12 hingga baris ke-15 yang merupakan perkalian matriks untuk membentuk sebuah matriks N antara A *transpose* dengan matriks A.

Kode Program 5. 7 Matriks N

No	Kode
1	`   A = []`
2	`   for item in tfIdf:`
3	`       temp = []`
4	`       for index in range(len(item)):`
5	`           if index > 0:`
6	`               temp.append(item[index])`
7	`       A.append(temp)`
8	`   A = np.asarray(A)`
9	`for item in A:`
10	`       print (item)`
11	`   At = A.transpose()`
12	`for item in At:`
13	`       print (item)`
14	`   AAt = np.matmul(At, A)`
15	`   for item in AAt:`
16	`       print (item)`

Pada Kode program 5.8 di baris ke-1 digunakan untuk mencari niali dari *eigenvalue.* Sedangkan pada varies ke-5 hingga ke-10 untuk mencari nilai dari akar lambda (λ) atau *eigenvalue* dan nilai *V* yang merupakan *eigenvector* yang digunakan untuk mengisi nilai matriks V. Pada kode program baris ke-16 hingga baris ke-24 digunakan untuk menyusun matriks diagonal ∑.

Kode Program 5. 8 Eigen Value dan Matriks S

No	Kode
1	`   eigenvalue, v = np.linalg.eigh(AAt)`
2	`   for item in eigenvalue:`
3	`       print (item)`
4	
5	`   akar_eigen = []`
6	`   for item in eigenvalue:`
7	`       if item > 0:`
8	`           akar_eigen.append(math.sqrt(item))`
9	`       else:`
10	`           akar_eigen.append(math.sqrt(item*(-`
11	`1)))`
12	
13	`   for item in akar_eigen:`
14	`       print (item)`

```
15
16      s = []
17      for x in range(len(akar_eigen)):
18          temp = []
19          for y in range(len(akar_eigen)):
20              if x == y:
21                  temp.append(akar_eigen[x])
22              else:
23                  temp.append(0)
24          s.append(temp)
```

Kode program 5.9 pada baris ke-1 samapai ke baris ke-4 digunakan untuk melakukan perhitungan invers dari matriks ∑. Invers matriks ∑ digunakan untuk menyusun matriks *U* pada tahapan selanjutnya. Sedangakan pada baris ke-7 digunakan untuk mencetak nilai dari matriks *V* yang didapatkan dari *eigenvector*. Serta pada baris program ke-10 digunakan untuk melakukan *transpose* nilai matriks *V* dari *eigenvector.* Pada baris ke-16 dan ke-17 merupakan perkalian matriks untuk mencari nilai dari matriks *U.*

Kode Program 5. 9 Matriks S *inverse* dan Matriks V

No	Kode
1	`s_inverse = np.linalg.inv(s)`
2	`print("\n## S Inverse##")`
3	`for item in s_inverse:`
4	`    print (item)`
5	
6	`print("\n## V ##")`
7	`for item in v:`
8	`    print (item)`
9	
10	`vt = v.transpose()`
11	`print("\n## v Transpose##")`
12	`for item in vt:`
13	`    print (item)`
14	
15	`print("\n##Hitung U##")`
16	`UU = np.matmul(v, s_inverse)`
17	`U = np.matmul(A, UU)`
18	`for item in U:`
19	`    print(item)`

Pada Kode program 5.10 di kode baris ke-2 hingga ke-7 merupakan kode yang digunakan untuk mencari nilai rata-rata pada baris matriks *vt* atau *V transpose*. Serta

pada baris ke-9 hingga ke-12 untuk melakukan filter pada nilai *vt* setelah diperoleh rata-rata dari setap baris matriksnya. Jika nilainya lebih dari rata-rata tersebut maka nilai pada matriks tidak diubah dan jik kurang dari nol maka akan diganti dengan nilai 0. Baris ke-16 hingga baris ke-21 digunakan untuk mencari nilai panjang atau *length* yang nantinya akan menentukan ringkasan dengan mengakarkan dari pangkat kuatdrat matriks kolom *vt* dengan nilai ∑. Serta baris ke-36 hingga baris ke-37 akan mencetak serta menampilkan urutan dari ringkasan berdasar besar nilai *length* serta menampilkan isi berita.

Kode Program 5. 10 Ekstraksi Ringkasan

No	Kode
1	`  avg = []`
2	`  for item in vt:`
3	`      temp = 0`
4	`      for bil in item:`
5	`          temp += bil`
6	`      temp = temp/len(item)`
7	`      avg.append(temp)`
8	
9	`  for x in range(len(vt)):`
10	`      for y in range(len(vt[x])):`
11	`          if vt[x][y] < avg[x]:`
12	`              vt[x][y] = 0`
13	
14	`  print("\n## length vts ##")`
15	`  vts=[]`
16	`  for x in range(len(vtvt)):`
17	`      total = 0`
18	`      for y in range(len(vtvt[x])):`
19	`          total += vtvt[y][x] * ss[y]`
20	`      total=math.sqrt(total)`
21	`      vts.append(total)`
22	
23	`  for item in vts:`
24	`      print(item)`
25	
26	`  total = vts`
27	`  vts = []`
28	`  for index in range(len(total)):`
29	`      temp = [index]`
30	`      temp.append(total[index])`
31	`      vts.append(temp)`
32	
33	`  sorted_vts = sorted(vts, key=itemgetter(1),`
34	`reverse=True)`

```
35      ##print("\n"+parsing[0])
36      for index in range(compression_rate):
37        print(parsing[sorted_vts[index][0]]+". ")
```