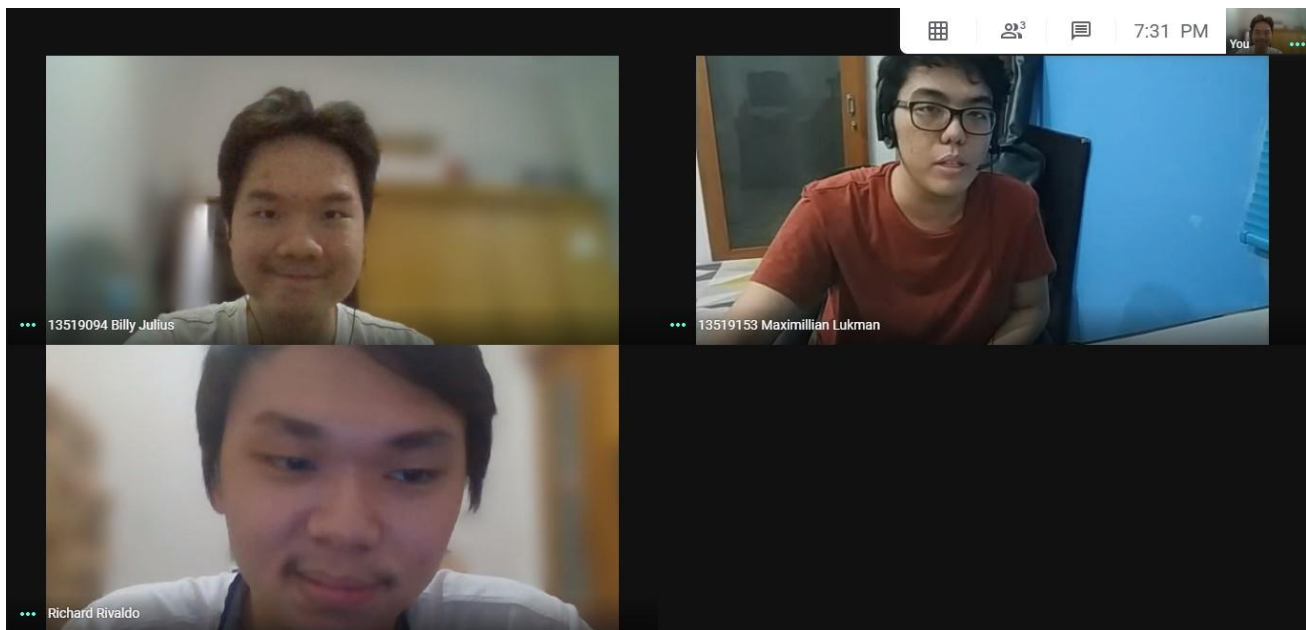


LAPORAN HASIL TUGAS BESAR 2
APLIKASI DOT PRODUCT PADA SISTEM TEMU-BALIK INFORMASI

DIAJUKAN UNTUK MEMENUHI TUGAS MATA KULIAH
IF 2123 – Aljabar Linier dan Geometri
SEMESTER I TAHUN 2020/2021

Disusun oleh :
Kelompok 43 / sabeb

Billy Julius	13519094
Maximillian Lukman	13519153
Richard Rivaldo	13519185



SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2020

DAFTAR ISI

BAB I DESKRIPSI MASALAH	2
BAB II TEORI SINGKAT	3
BAB III IMPLEMENTASI PROGRAM	6
BAB IV EKSPERIMEN	23
BAB V SIMPULAN, SARAN, DAN REFLEKSI.....	31
DAFTAR REFERENSI.....	33

BAB I

DESKRIPSI MASALAH

Buatlah program mesin pencarian dengan sebuah website lokal sederhana. Spesifikasi program adalah sebagai berikut:

1. Program mampu menerima search query. Search query dapat berupa kata dasar maupun berimbuhan.
2. Dokumen yang akan menjadi kandidat dibebaskan formatnya dan disiapkan secara manual. Minimal terdapat 15 dokumen berbeda sebagai kandidat dokumen.
Bonus: Gunakan web scraping untuk mengekstraksi dokumen dari website.
3. Hasil pencarian yang terurut berdasarkan similaritas tertinggi dari hasil teratas hingga hasil terbawah berupa judul dokumen dan kalimat pertama dari dokumen tersebut. Sertakan juga nilai similaritas tiap dokumen.
4. Program disarankan untuk melakukan pembersihan dokumen terlebih dahulu sebelum diproses dalam perhitungan cosine similarity. Pembersihan dokumen bisa meliputi hal-hal berikut ini.
 - a. Stemming dan penghapusan stopwords dari isi dokumen.
 - b. Penghapusan karakter-karakter yang tidak perlu.
5. Program dibuat dalam sebuah website lokal sederhana. Dibebaskan untuk menggunakan framework pemrograman website apapun. Salah satu framework website yang bisa dimanfaatkan adalah Flask (Python), ReactJS, dan PHP.
6. Kalian dapat menambahkan fitur fungsional lain yang menunjang program yang anda buat (unsur kreativitas diperbolehkan/dianjurkan).
7. Program harus modular dan mengandung komentar yang jelas.
8. Dilarang menggunakan library cosine similarity yang sudah jadi.

BAB II

TEORI SINGKAT

Information Retrieval

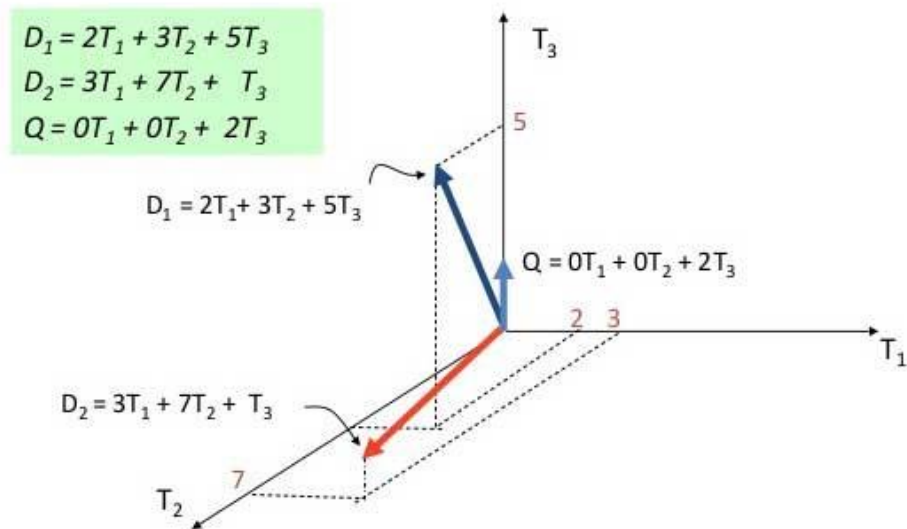
Information retrieval atau sistem temu balik adalah menemukan(biasanya dokumen) dari sebuah ketidakstrukturan yang alami(biasanya teks) untuk memenuhi sebuah kebutuhan informasi dari koleksi yang berukuran besar(biasanya disimpan pada komputer). Sistem information retrieval atau sistem temu balik informasi bertujuan untuk mencukupi kebutuhan informasi pengguna dengan sumber informasi yang tersedia sesuai dengan situasi. Penulis mempresentasikan ide dan pikiran mereka ke dalam sebuah dokumen, pencari dokumen mencari sebuah dokumen di dalam sekumpulan dokumen dimana pencari tersebut tidak mengetahui dengan pasti bagaimana cara menemukan dan mengenali dokumen yang tepat sesuai dengan kebutuhannya, sistem temu balik informasi mempertemukan ide yang ditulis penulis tersebut dengan kebutuhan informasi yang dibutuhkan oleh pencari dokumen tersebut yang dinyatakan kedalam pernyataan (query).

Sistem temu balik informasi (*information retrieval*) ini digunakan untuk menemukan kembali informasi-informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis. Salah satu aplikasi umum dari sistem temu kembali informasi adalah search-engine atau mesin pencarian yang terdapat pada jaringan internet. Pengguna dapat mencari halaman-halaman Web yang dibutuhkannya melalui mesin tersebut.

Model Ruang Vektor

Kemiripan kalimat memainkan peran penting pada berbagai penelitian yang berhubungan dengan teks dan aplikasi. Model Ruang Vektor digunakan sebagai representasi dari kumpulan dataset dokumen teks.

Dokumen dalam Model Ruang Vektor berupa matriks yang berisi bobot seluruh kata pada tiap dokumen. Bobot tersebut menyatakan kepentingan atau kontribusi kata terhadap suatu dokumen dan kumpulan dokumen. Kepentingan suatu kata dalam dokumen dapat dilihat dari frekuensi kemunculannya terhadap dokumen.



Gambar diatas menunjukkan pemodelan dokumen teks di ruang dimensi dimana **(D)** adalah kalimat dokumen sedangkan **(T)** adalah term atau kata.

Dalam model ruang vektor, dokumen dan query direpresentasikan sebagai vektor dalam dalam ruang vektor yang disusun dalam indeks term, kemudian dimodelkan dengan persamaan geometri. Sedangkan model probabilistik membuat asumsi-asumsi distribusi term dalam dokumen relevan dan tidak relevan dalam orde estimasi kemungkinan relevansi suatu dokumen terhadap suatu query.

Cosine Similarity

Cosine Similarity dapat diimplementasikan untuk menghitung nilai kemiripan antar kalimat dan menjadi salah satu teknik untuk mengukur kemiripan teks yang populer, yaitu dengan menghitung sudut antara vektor dokumen dengan vektor kueri. Jika vektor

adalah satuan panjang, cosinus dari sudut antara mereka hanyalah dot product dari vektor, persamaannya sebagai berikut:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

Dimana :

A : vektor A

B : vektor B

|A| : besar dari vektor A

|B| : besar dari vektor B

Karena berdasarkan cosinus sudut antara dua vektor, maka nilainya berkisar pada 0 sampai dengan 1, dimana 0 menandakan bahwa kedua dokumen tidak mirip sama sekali, dan 1 menandakan bahwa antar dokumen benar-benar identik.

BAB III

IMPLEMENTASI PROGRAM

Implementasi program dalam tugas besar ini kami bagi menjadi 2 bagian besar. Bagian pertama adalah bagian *front-end* yang merupakan tempat dimana kami merealisasikan hampir keseluruhan struktur dan tampilan website yang kami gunakan untuk *search-engine* kami seperti file html dan css. Kemudian bagian kedua adalah bagian *back-end* dan dikhususkan untuk menyimpan realisasi dari program-program yang digunakan untuk *search-engine* dimana kami menggunakan bahasa Python sebagai *back-end* kami.

Pada bagian *back-end* terdiri dari beberapa *file* program, ***WebScrap.py***, ***Read.py***, ***Preprocessing.py***, ***form.py***, dan ***Main.py***. Pada file pertama yaitu ***WebScrap.py***, kami menggunakan library BeautifulSoup untuk memudahkan proses *web scraping* lalu terdapat dua fungsi atau *method* utama dalam file ini yaitu:

```
def Convert(string):  
    list1=[]  
    list1[:0]=string  
    return list1
```

Ini adalah fungsi **Convert** yang merupakan fungsi konversi biasa dan berguna untuk mengonversi string menjadi list of char untuk memudahkan pengecekan.

Lalu yang kedua adalah fungsi utama untuk melakukan *web scraping* dimana kami memanfaatkan library BeautifulSoup tadi yang sudah diimport dan membuat web request khusus terhadap dua halaman pertama dari web *reuters* yang akan kami ambil setiap link artikelnya. Dalam setiap artikel, kami menentukan kalimat pertamanya, menentukan kontennya lalu menghilangkan format-format yang tidak diinginkan termasuk format penamaan file .txt-nya sehingga semua artikelnya dapat diexport.

```

def webScrap() :
    # membuat web request untuk 2 halaman pertama dari website reuters
    page = requests.get('https://www.reuters.com/news/archive/indonesia').text
    page2 = requests.get('https://www.reuters.com/news/archive/indonesia?view-page&page=2&pageSize=10').text
    # mendefinisikan BeautifulSoup untuk masing-masing halaman
    soup = BeautifulSoup(page, 'lxml')
    soup2 = BeautifulSoup(page2, 'lxml')

    TLink = [] # variabel penampung semua link
    # untuk tiap class 'story', diambil link a href pertama yang ditemukan, kemudian diappend ke variabel TLink
    for list in soup.find_all('article', class_='story') :
        link = list.find('a')
        linkh = link['href']
        TLink.append('https://www.reuters.com'+linkh)
    # ulangi untuk halaman kedua
    for list in soup2.find_all('article', class_='story') :
        link = list.find('a')
        linkh = link['href']
        TLink.append('https://www.reuters.com'+linkh)

    # tinjau satu-satu link yang ada
    for linki in TLink :
        # buat web req dan definisikan BeautifulSoup untuk masing-masing link
        pagei = requests.get(linki).text
        soupi = BeautifulSoup(pagei, 'lxml')

        content = [] # variabel penampung konten paragraf
        # mencari headline
        for h1 in soupi.find_all('h1') :
            headline = h1.text
            # Pengubahan untuk keperluan pencarian kalimat pertama yang lancar nantinya
            headline = headline.replace("U.S." , "US")
        # mencari paragraf yang ditandai class 'Paragraph-paragraph-2Bgue ArticleBody-para-TD_9x' sebagai konten
        for p in soupi.find_all('p', class_='Paragraph-paragraph-2Bgue ArticleBody-para-TD_9x'):
            paragraph = p.text
            # Pengubahan untuk keperluan pencarian kalimat pertama yang lancar nantinya
            paragraph = paragraph.replace("U.S." , "US")
            paragraph = paragraph.replace("Del." , "Del")
            content.append(paragraph)

        # sedikit perbaikan terhadap judul yang tidak bisa dijadikan nama file
        headlist = Convert(headline)
        for i in range(len(headlist)) :
            if headlist[i] == ':' :
                headlist[i] = ''
            if headlist[i] == '?' :
                headlist[i] = ''
        headline = ''.join(headlist)

        # export file dengan nama headline dan konten content
        exportFile = open('../test/'+headline+'.txt',"w", encoding='utf-8')
        exportFile.write(' '.join(content))
        exportFile.close()

```


File program yang kedua untuk bagian *back-end* adalah **Read.py**, pada file ini kami menggunakan library *nltk* untuk memudahkan prosesnya. Fungsi-fungsi dalam file ini digunakan untuk mengambil isi dari setiap file txt yang sudah di *scraping* menggunakan *WebScrap.py* sebelumnya. Fungsi-fungsi tersebut adalah:

Fungsi yang pertama adalah **readFile** yang digunakan untuk membuka dan membaca file-file txt dan mengganti *newline* dengan spasi serta mengecilkan semua huruf besar untuk memudahkan pencarian.

```
def readfile(namafile):  
    # Open File  
    with open(namafile, 'r', encoding='utf-8') as File:  
        # Replace newlines with space  
        contents = File.read().replace("\n", " ")  
  
        # Change the texts into lowercase  
        contents = contents.lower()  
  
    return contents
```

Lalu fungsi yang kedua adalah **cleaning** dimana fungsi ini menerima masukan sebuah string dan akan di'bersih'kan dari segala tanda baca dan *special characters* dengan memanfaatkan library *nltk* yang sudah diimport sebelumnya.

```
def cleaning(strings):  
    # Can be used to remove HTML Tags for Web Scraping  
    # Replace with Whitespace, same length with the punctuation  
    punc = str.maketrans(string.punctuation, ' '*len(string.punctuation))  
    return (strings.translate(punc))
```

Fungsi yang terakhir pada file *Read.py* adalah fungsi **token** yang berguna untuk mengubah kalimat string menjadi sebuah list of string yang dipisahkan per-karakter.

```
def token(strings):  
    # Tokenize the strings into a list containing words  
    return (nlk.word_tokenize(strings))
```

Lalu pada file program yang ketiga pada *back-end* yaitu **Preprocessing.py** terdapat dua fungsi/method yang digunakan untuk melakukan proses terhadap query dan content dari artikel yang sudah dibaca. Pada file ini kami juga menggunakan library nltk untuk memudahkan proses penghilangan stopwords dan *stemming*.

Fungsi yang pertama pada file ini adalah untuk menghilangkan **stopwords**. Stopword merupakan kata yang diabaikan dalam pemrosesan, kata-kata ini biasanya adalah kata yang mempunyai frekuensi kemunculan yang tinggi misalnya seperti kata penghubung. Kami memanfaatkan library nltk untuk fungsi stopwords ini.

```
# Remove stopwords using stopwords corpus in NLTK as database

def stopwords(Tokens):
    return([i for i in Tokens if i not in nltk.corpus.stopwords.words("english")])
```

Yang kedua adalah fungsi **stemming** dimana fungsi ini berguna untuk mengubah setiap kata menjadi '*root-form*' atau bentuk dasarnya. Dengan adanya dua fungsi ini yaitu menghilangkan stopwords dan *stemming*, proses pencarian antara query dengan content dari artikel akan lebih mudah.

```
def stemming(Tokens):
    # Create an object of Porter Stemmer
    Porter_Stemmer = nltk.stem.PorterStemmer()

    # Saving the stemmed version of each word into a list
    Stemmed = list()
    for words in Tokens:
        Stemmed.append(Porter_Stemmer.stem(words))
    return(Stemmed)
```

Pada file program yang keempat yaitu **form.py**, terdapat method menggunakan flask untuk mendapatkan class beserta komponen-komponennya yang dibutuhkan untuk membuat query form dan juga upload form untuk ditampilkan di website *search-engine* kami.

```

from flask_wtf import FlaskForm
from wtforms import StringField , SubmitField , MultipleFileField
from wtforms.validators import DataRequired

class QueryForm(FlaskForm) :
    query = StringField('Query' , validators=[DataRequired()])
    submit = SubmitField('Search')

class UploadForm(FlaskForm) :
    file = MultipleFileField('File')
    submit = SubmitField('Submit')

```

File yang terakhir untuk *back-end* adalah drivernya yaitu **Main.py**, disini seluruh program yang telah direalisasikan akan dipanggil dan dieksekusi

```

from flask import Flask , render_template, url_for , flash , redirect , request
from werkzeug.utils import secure_filename
import os
import Read
import Preprocessing
import math
from form import QueryForm , UploadForm

```

Ini adalah library-library yang diperlukan dan diimport ke dalam driver Main.py.

```

fileList = []
for root, dirs, files in os.walk('../test', topdown=False):
    for name in files :
        dir = os.path.join(root,name).split('\\') # Mengambil hanya nama filenya,
        oleh \
        fileList.append(dir[1])

contentList = []
for name in fileList :
    # Menampung tiap konten dalam tiap file, melakukan cleaning, konversi ke token
    content = Read.readfile('../test/'+name)
    content = Read.cleaning(content)
    content = Read.token(content)
    content = Preprocessing.stopwords(content)
    content = Preprocessing.stemming(content)

    # Menggabungkan semua konten menjadi satu list
    contentList.append(content)

```

Pertama-tama semua file yang telah di-*web-scraping* ditampung kedalam sebuah list yang kemudian ditampung setiap konteks dalam setiap file dengan terlebih dahulu menghilangkan *special characters*, dikonversi menjadi token(list of string), menghilangkan stopwords, dan dicari bentuk dasarnya dari setiap kata.

```
wordList = []
for content in contentList :
    for word in content :
        if word not in wordList :
            wordList.append(word)

# Membuat variabel penampung jumlah kemunculan tiap kata pada tiap data
mVec = [[0 for x in range(len(wordList))] for y in range(len(fileList))]
j = 0
for content in contentList :
    for word in content :
        for i in range(len(wordList)) :
            if word == wordList[i] :
                mVec[j][i] = mVec[j][i] + 1
    j = j + 1
```

Lalu setiap artikel/content yang sudah di-*tokenize* atau menjadi list of string ditampung setiap katanya ke dalam list wordList yang kemudian setiap wordnya dihitung banyak kemunculannya di setiap dokumen/artikel dengan cara mencocokkan dengan list wordList yang berisi kata dari setiap dokumen/artikel tersebut.

Kemudian ada method baru yaitu **Search** sebagai fungsi utama yang menerima masukan query dan memrosesnya dan menentukan apakah sebuah query kosong atau bukan lalu menghilangkan kata yang berulang pada query sekaligus dihitung berapa kali kemunculannya. Lalu fungsi juga menghitung dot product, similaritas, dan menginisialisasi data-data yang diperlukan untuk hasil pencarian query seperti judul-judul dokumen, kalimat pertama dan juga perhitungan similarity dari tiap-tiap dokumen kemudian mengurutkannya sesuai tingkat similaritasnya. Penjelasan lebih lengkapnya dapat dilihat pada source code Main.py

```

def Search(query) :
    global fileList , wordList , mVec

    # Terima query, lakukan pembersihan dll ke query seperti ke database
    query = Read.cleaning(query)
    query = Read.token(query)
    query = Preprocessing.stopwords(query)
    query = Preprocessing.stemming(query)

    # query = ['red','big','car','red','car']

    if (query != []) :
        # Proses menghasilkan vektor q yang sudah dihilangkan kata berulang
        # q2 = ['red','big','car']
        q2 = []
        for word in query :
            if word not in q2 :
                q2.append(word)
        # num_q = [2,1,2]
        num_q = [0 for x in range(len(q2))]
        for word in query :
            for i in range(len(q2)) :
                if word == q2[i] :
                    num_q[i] += 1

        qWord = []
        # variabel untuk menampung kata yang sudah di proses
        dump = []
        for word in query :
            # tidak akan memproses kata yang sudah di proses
            if word not in dump :
                for i in range(len(wordList)) :
                    if word == wordList[i] :
                        qWord.append(i)
                dump.append(word)

        # Query bisa berulang, jadi dicatat juga kemunculan tiap kata di query yang muncul di database
        num_qWord =[0 for x in range(len(qWord))]
        for word in query :
            for i in range(len(qWord)) :
                if word == wordList[qWord[i]]:
                    num_qWord[i] += 1

        # Menghitung similaritas
        sim = []

        # Inisiasi tabel untuk pembuatan tabel term
        T = [[0 for i in range(len(fileList) + 2)] for j in range(len(q2))]
        # Dua kolom pertama diisi oleh term dan jumlah kemunculannya pada query
        for i in range(len(q2)) :
            T[i][0] = q2[i]
            T[i][1] = num_q[i]

```

```

for i in range(len(fileList)) :
    # Menghitung Dot Product
    dotProd = 0
    k = 0
    for idx in qWord :
        dotProd += mVec[i][idx] * num_qWord[k] # num_qWord adalah jlh kemunculan pada
        kemunculan pada data
        k +=1

    # Menghitung norma dari query
    sumQ = 0
    for l in num_q :
        sumQ += pow(l,2)
    normQ = float(math.sqrt(sumQ))

    # Menghitung norma dari data
    sumD = 0
    for j in range(len(wordList)) :
        sumD += pow(mVec[i][j],2)
    normD = float(math.sqrt(sumD))

    # Menghitung similaritas data sekarang dengan query, dan dimasukkan ke list similar
    currSim = float(dotProd/(normQ * normD))
    sim.append(currSim)

# Menghasilkan list yang menampung jumlah kemunculan tiap query pada database untuk
tabel term
inc = 0
for word in q2 :
    for inc2 in range(len(wordList)) :
        if wordList[inc2] == word :
            T[inc][i+2] = mVec[i][inc2]
        inc += 1

# List untuk header tiap kolom
THeader = [0 for i in range(len(fileList) + 2)]
THeader[0] = 'Term'
THeader[1] = 'Query'

for i in range(len(fileList)) :
    THeader[i+2] = "D"+str(i+1)

else :
    T = []
    THeader = []
    sim = [0 for x in range(len(fileList))]

```

```

# Membuat array yang menampung judul dokumen dengan format yang telah dislice
titleList = [Title[:-4] for Title in fileList]

# Membuat array yang menampung kalimat pertama dari setiap data
headList = []
for titles in fileList:
    with open('../test/'+titles, encoding='utf-8') as f:
        head = f.read().split('.')
        headList.append(head[0] + '.')

# Menginisialisasi Array 2D berisi perhitungan similarity, judul dokumen, dan kalimat pertama
processedFiles = [[0 for i in range(3)] for j in range(len(fileList))]

# Memasukkan tingkat similarity tiap dokumen
for i in range(len(sim)):
    processedFiles[i][0] = sim[i]

# Memasukkan semua judul dokumen
for i in range(len(fileList)):
    processedFiles[i][1] = titleList[i]

# Memasukkan kalimat pertama dari setiap dokumen
for i in range(len(headList)):
    processedFiles[i][2] = headList[i]

# Mengurutkan array yang telah didapat dengan key berupa kolom pertama, yaitu tingkat similarit
# dari yang paling tinggi
sortedProcessed = sorted(processedFiles, key = lambda keyCol:keyCol[0], reverse=True)

# Menginisialisasi list kosong untuk menampung proses selanjutnya
processedLD = []

# Memasukkan tiap atribut yang ada ke dalam sebuah dictionary dengan key berupa Similarity, Tit
FirstSentence, ID, dan Words
for articles in range(len(sortedProcessed)):
    with open('../test/'+sortedProcessed[articles][1]+'.txt'), 'r', encoding='utf-8') as File:
        contents = File.read()
        count = len(contents.split(" "))
        attributeDict = {"Similarity" : sortedProcessed[articles][0],
                        "Title" : sortedProcessed[articles][1],
                        "FirstSentence" : sortedProcessed[articles][2],
                        "Contents" : contents,
                        "ID" : "Doc" + str(articles+1),
                        "Words" : count
                        }
        # Memasukkan tiap dictionary yang telah dihasilkan ke dalam list sebelumnya
        processedLD.append(attributeDict)

return processedLD , T , THeader

```

Ada juga method **renderDocList** yang berguna untuk menampilkan/me-render seluruh dokumen sebagai Side List di website dan diurutkan berdasarkan abjad.

```
def renderDocList() :  
    global fileList  
  
    # Menginisialisasi list kosong untuk menampung judul dari tiap dokumen  
    docTitle = []  
  
    # Memasukkan tiap judul dokumen ke dalam dictionary dengan atribut judul  
    for name in fileList :  
        titleDict = {"Title" : name}  
        docTitle.append(titleDict)  
  
    return docTitle
```

Code lain yang ada di dalam Main.py adalah menampilkan hasil perhitungan, tabel, dan render halaman website menggunakan Flask

Pertama adalah menampilkan primary page yaitu search page yang menampilkan semua dokumen yang ada sebagai Side List, search box, tabel term dan query, dan artikel-artikel hasil pencarian berikut dengan similaritynya.

```
# Rute utama, utk query dan hasil searching  
@app.route('/', methods=['GET', 'POST'])  
def query():  
    # Ambil variabel global postQ, T, THeader, docList  
    global postQ, T, THeader, docList  
    postQ = []  
    T = []  
    THeader = []  
    # Karena ada kemungkinan upload baru, docList selalu di-update  
    docList = renderDocList()  
    # Panggil QueryForm dari form.py  
    form = QueryForm()  
    # Jika ada query yang disubmit, lakukan proses search, dan oper ke query.html berupa list similarity, list tiap baris table, dan list header tabel  
    if form.validate_on_submit() :  
        flash(f'Searching succeeded!', 'success')  
        postQ, T, THeader = Search(form.query.data)  
        return render_template('query.html', posts = postQ, docs = docList, T = T, THeader = THeader, form = form)  
    # Jika tidak ada (pertama kali), hasil searching tampilkan kosong  
    return render_template('query.html', posts = postQ, docs = docList, T = T, THeader = THeader, form = form)
```


Kedua adalah menampilkan untuk saat kita membuka salah satu link dari artikel hasil pencarian.

```
# Rute untuk tiap artikel
@app.route('/article/<id>')
def article(id):
    # Ambil variabel global postQ , oper ke article.html untuk diproses pengecekan id mana
    global postQ
    return render_template("article.html" , id = id, posts = postQ, title = "Article")
```

Yang ketiga adalah untuk menampilkan page untuk upload file ke dalam website.

```
# Rute untuk upload file
@app.route('/upload', methods=['GET', 'POST'])
def upload():
    # Panggil UploadForm dari form.py
    form = UploadForm()
    global fileList , wordList , mVec

    # Jika ada file tersubmit, ambil nama file (secured oleh werkzeug) , cek ekstensinya jika .txt, simpan file ke
    folder test
    if form.validate_on_submit():
        for files in form.file.data :
            filename = secure_filename(files.filename)
            file_ext = os.path.splitext(filename)[1]
            if file_ext != '.txt':
                flash(f'Format of file(s) uploaded is not allowed (.txt only), file submission canceled!' , 'danger')
                return redirect(url_for('upload'))
            files.save('../test/' + filename)
        flash(f'File(s) added!' , 'success')

    # Karena ada input file baru, database kamus di-update
    # Menampung semua nama file ke dalam suatu variabel list fileList
    fileList = []
    for root, dirs, files in os.walk('../test', topdown=False):
        for name in files :
            dir = os.path.join(root,name).split('\\') # Mengambil hanya nama filenya, tidak bersama direktori yan
            displit oleh \
            fileList.append(dir[1])

    contentList = []
    for name in fileList :
        # Menampung tiap konten dalam tiap file, melakukan cleaning, konversi ke token, menghapus stopwords
        lemmatize
        content = Read.readfile('../test/'+name)
        content = Read.cleaning(content)
        content = Read.token(content)
        content = Preprocessing.stopwords(content)
        content = Preprocessing.stemming(content)

        # Menggabungkan semua konten menjadi satu list
        contentList.append(content)
```

```

# Membuat variabel penampung kata-kata yang ada di dokumen
wordList = []
for content in contentList :
    for word in content :
        if word not in wordList :
            wordList.append(word)

# Membuat variabel penampung jumlah kemunculan tiap kata pada tiap data
mVec = [[0 for x in range(len(wordList))] for y in range(len(fileList))]
j = 0
for content in contentList :
    for word in content :
        for i in range(len(wordList)) :
            if word == wordList[i] :
                mVec[j][i] = mVec[j][i] + 1
        j = j + 1

return redirect(url_for('upload'))
return render_template('upload.html', form=form)

```

Pada bagian kedua *front-end* kami membagi file html menjadi 4 bagian yaitu untuk layout utama, page untuk search/memasukkan query, menampilkan artikel yang dipilih, dan page untuk upload file local ke website.

File pertama adalah **layout.html** yang dijadikan file html utama untuk tampilan awal.

```

<!DOCTYPE html>
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
        integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

        <link rel="stylesheet" type="text/css" href="{{ url_for('static' , filename = 'main.css') }}">

        {% if title %}
        <title>Search Engine - {{title}}</title>
        {% else %}
        <title>Search Engine - sabeb</title>
        {% endif %}
    </head>

```

Dapat dilihat diatas adalah Bootstrap yang digunakan untuk css dan juga file css yaitu main.css

```

<body>
  <header class="site-header">
    <nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">
      <div class="container">
        <a class="navbar-brand mr-4" href="/">sabeb</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarToggle"
          aria-controls="navbarToggle" aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarToggle">
          <div class="navbar-nav mr-auto">
            <a class="nav-item nav-link" href="{{ url_for('query') }}">Search</a>
            <a class="nav-item nav-link" href="{{ url_for('upload') }}">Upload</a>
          </div>
        </div>
      </div>
    </nav>
  </header>

  <main role="main" class="container">
    <div class="row">

    </div>
    <div class="row">
      <div class="col-md-8">
        {% with messages = get_flashed_messages(with_categories=true) %}
          {% if messages %}
            {% for category,message in messages %}
              <div class="alert alert-{{category}}">
                {{message}}
              </div>
            {% endfor %}
          {% endif %}
        {% endwith %}
        {% block table %}{% endblock %}
        {% block query %}{% endblock %}
        {% block content %}{% endblock %}
      </div>
      {% block sideList %}{% endblock %}
    </div>
  </main>

  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965Dz00rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/umd/popper.min.js" integrity="sha384-U02eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1cLHTMga3JDZwrnQq4sF86dIHNDz0W1" crossorigin="anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
</body>
</html>

```

Terdapat beberapa block pada tag <main> yang didapat dari file html lainnya seperti block table, query, content, dan sideList.

Lalu, file kedua **query.html** adalah untuk menampilkan page search atau query serta hasil dari pencarian query tersebut.

```
{% extends "layout.html" %}

{% block content %}
    {% for post in posts %}
        <article class="media content-section">
            <div class="media-body">
                <div class="article-metadata">
                    <p class='text-muted'>Similarity : {{ post.Similarity * 100 }} %</p>
                    <p class='text-muted'>Total Words : {{ post.Words }}</p>
                </div>
                <h2><a class="article-title" href="/article/{{ post.ID }}">{{ post.Title }}</a></h2>
                <p class="article-content">{{ post.FirstSentence }}</p>
            </div>
        </article>
    {% endfor %}
{% endblock content %}

{% block query %}
    <div class="content-section">
        <form method="POST" action="">
            {{ form.hidden_tag() }}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Search</legend>
                <div class = "form-group">
                    {{ form.query.label(class="form-control-label") }}
                    {% if form.query.errors %}
                        {{ form.query(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.query.errors %}
                                <span>{{ error }} </span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.query(class="form-control form-control-lg") }}
                    {% endif %}
                </div>
            </fieldset>
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-info") }}
            </div>
        </form>
    </div>
{% endblock query %}
```

block content adalah untuk menampilkan setiap artikel berikut dengan similaritasnya berdasarkan query yang didapat, sedangkan block query adalah untuk membuat form search untuk querynya.

```

{% block sideList %}
<div class="col-md-4">
  <div class="content-section">
    <h3>Side List</h3>
    <p class='text-muted'>Here is a list of documents used
    <article class="media content-section">
      <div class="media-body">
        <div class="article-metadata">
          Documents used :
        </div>
        {% for post in docs %}
          <p>{{ post.Title }}</p>
        {% endfor %}
      </div>
    </article>
  </p>
</div>
</div>
{% endblock sideList %}

{% block table %}
{% if T != [] %}
<div class="content-section">
  <div style="overflow-x:auto;">
    <table id="term-table">
      <thead>
        {% for column in THeader %}
          <th>{{ column }}</th>
        {% endfor %}
      </thead>
      <tbody>
        {% for row in T %}
          <tr>
            {% for data in row %}
              <td>{{ data }}</td>
            {% endfor %}
          </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>
</div>
{% endif %}
{% endblock %}

```

block sideList untuk menampilkan daftar semua dokumen di sebelah kanan dan block table digunakan saat query diproses dan menampilkan tabel term and query dan kemunculannya di setiap dokumen.

Sedangkan pada file **article.html** hanya untuk page artikel/dokumen yang ditampilkan dan juga similaritas dari artikel tersebut terhadap hasil pencarian query.

```
{% extends "layout.html" %}
{% block content %}
    {% for post in posts %}
        {% if id == post.ID %}
            <article class="media content-section">
                <div class="media-body">
                    <div class="article-metadata">
                        <p class='text-muted'>Similarity : {{ post.Similarity * 100 }} %</p>
                        <p class='text-muted'>Total Words : {{ post.Words }}</p>
                    </div>
                    <h2><a class="article-title">{{ post.Title }}</a></h2>
                    <p class="article-content">{{ post.Contents }}</p>
                </div>
            </article>
        {% endif %}
    {% endfor %}
{% endblock content %}
```

Dan terakhir pada file **upload.html** dikhususkan untuk block-block page upload dimana dibutuhkan form agar dapat mengunggah file atau dokumen baru.

```
{% extends "layout.html" %}
{% block query %}
    <div class="content-section">
        <form method="POST" action="" enctype="multipart/form-data">
            {{ form.hidden_tag() }}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Upload File</legend>
                <div class="form-group">
                    {{ form.file.label(class="form-control-label") }}
                    {% if form.file.errors %}
                        {{ form.file(class="form-control form-control-lg is-invalid") }}
                        <div class="invalid-feedback">
                            {% for error in form.file.errors %}
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% else %}
                        {{ form.file(class="form-control form-control-lg") }}
                    {% endif %}
                </div>
            </fieldset>
            <div class="form-group">
                {{ form.submit(class="btn btn-outline-info") }}
            </div>
        </form>
    </div>
{% endblock query %}
```

```

{% block sideList %}
<div class="col-md-4">
  <div class="content-section">
    <h3>Side List</h3>
    <p class='text-muted'>Here is a list of documents used
    <article class="media content-section">
      <div class="media-body">
        <div class="article-metadata">
          Documents used :
        </div>
        {% for post in docs %}
          <p>{{ post.Title }}</p>
        {% endfor %}
        </div>
      </article>
    </p>
  </div>
</div>
{% endblock sideList %}

```

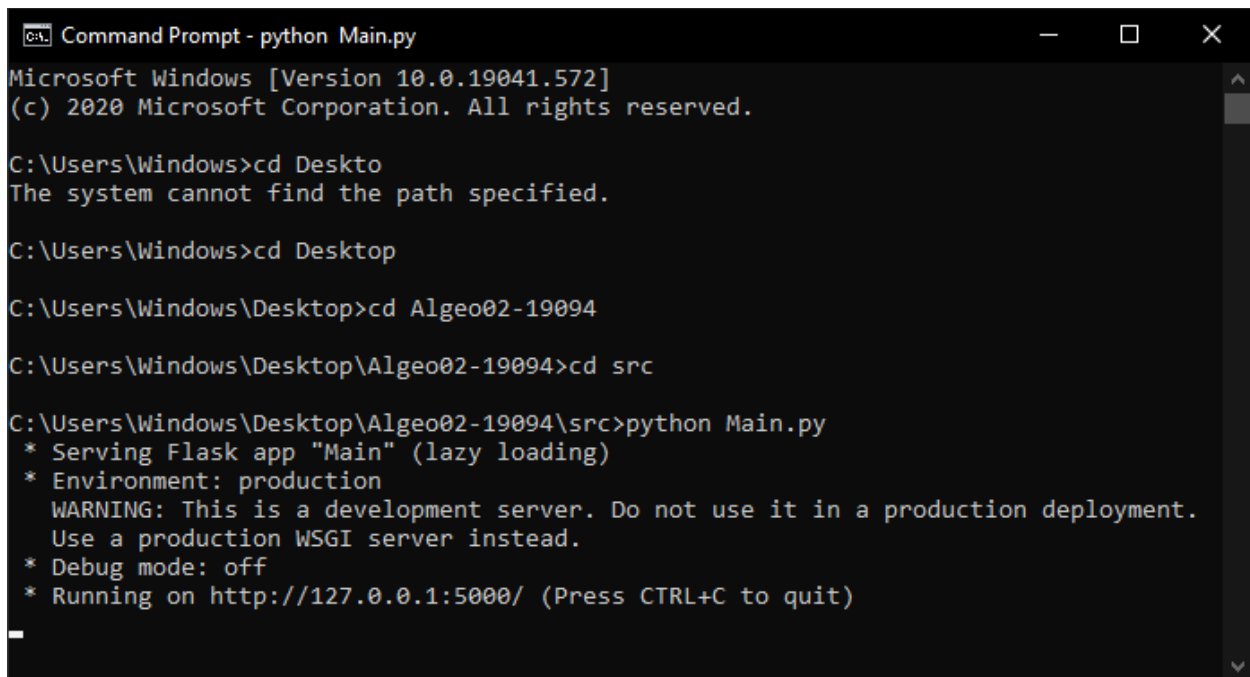
File-file html ini disambungkan dengan back-end melalui Flask dengan mengoper atau mendapatkan nilai/value dari back-end hasil run Main.py dan menampilkannya di page sesuai dengan '@route'-nya: query/search, upload, menampilkan artikel.

BAB IV

EKSPERIMEN

Berikut ini adalah beberapa hasil program kami terhadap beberapa *test-case* yang kami uji cobakan seperti mendapatkan input kosong, mendapatkan input angka, input stopwords, input kalimat biasa, dan input kalimat yang dicampur dengan angka atau simbol.

Setelah program driver utama yaitu Main.py di run,



```
Command Prompt - python Main.py
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Windows>cd Deskto
The system cannot find the path specified.

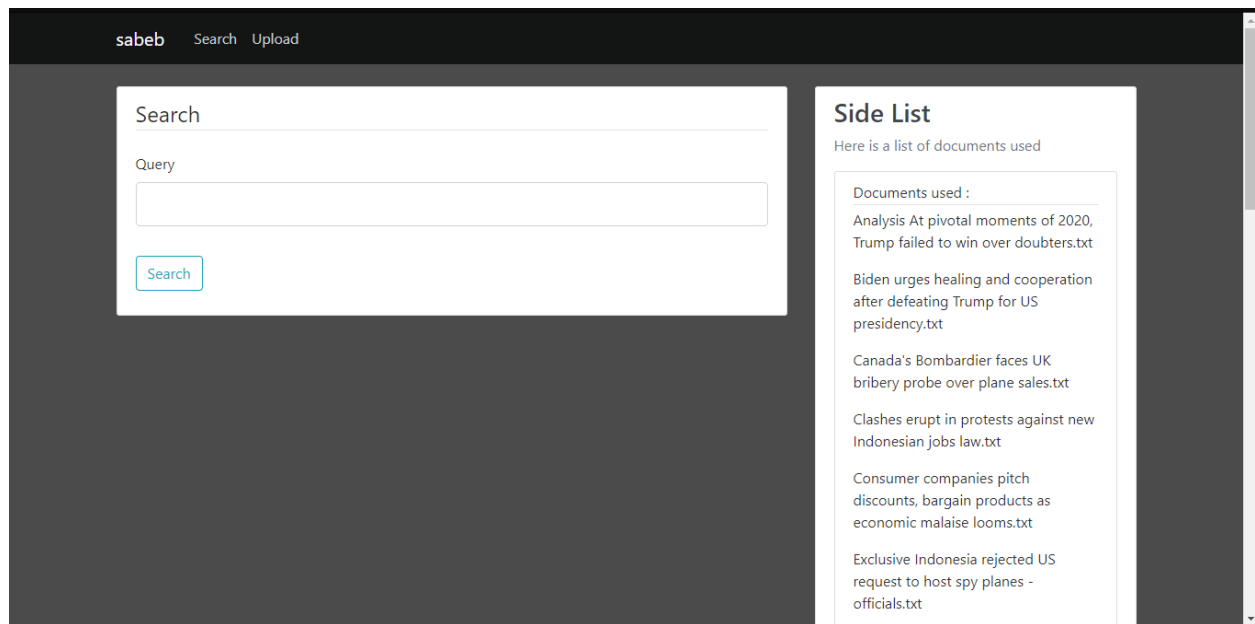
C:\Users\Windows>cd Desktop

C:\Users\Windows\Desktop>cd Algeo02-19094

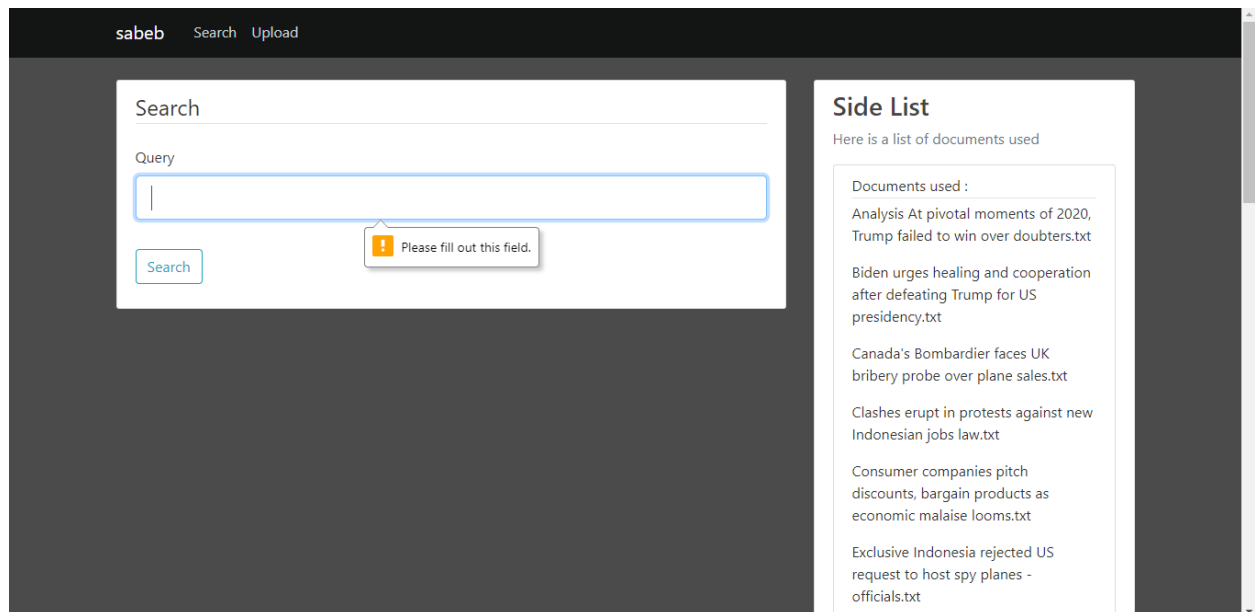
C:\Users\Windows\Desktop\Algeo02-19094>cd src

C:\Users\Windows\Desktop\Algeo02-19094\src>python Main.py
* Serving Flask app "Main" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Website search engine dapat di akses melalui link yang di-generate yaitu dalam contoh gambar di atas **<http://127.0.0.1:5000/>** dan membukanya di salah satu browser. Setelah itu akan langsung muncul tampilan awal seperti dibawah ini.



Setelah itu, dicoba *test-case* saat input query kosong, hasilnya adalah



Akan ada notifikasi bahwa bagian search box harus diisi dan tidak akan menampilkan hasil artikel apapun.

Lalu ada juga *test-case* saat mendapatkan input angka adalah sebagai berikut

sabebSearchUpload

Searching succeeded!

Term	Query	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
1	1	0	0	1	1	2	0	0	2	2	0	1	1	0	1
2	1	0	1	0	0	2	0	0	0	0	0	0	0	0	0

Search

Query

1 2

Search

Similarity : 8.455318085801517 %

Side List

Here is a list of documents used

Documents used :

Analysis At pivotal moments of 2020, Trump failed to win over doubters.txt

Biden urges healing and cooperation after defeating Trump for US presidency.txt

Canada's Bombardier faces UK bribery probe over plane sales.txt

Clashes erupt in protests against new Indonesian jobs law.txt

Consumer companies pitch discounts, bargain products as economic malaise looms.txt

Exclusive Indonesia rejected US request to host spy planes - officials.txt

Search

Query

1 2

Search

Similarity : 8.455318085801517 %

Total Words : 630

Consumer companies pitch discounts, bargain products as economic malaise looms

(Reuters) - Unilever, Procter & Gamble and other major consumer goods manufacturers are

Pada gambar diatas, artikel dapat dicari dengan input/query angka.

Selanjutnya ada *test-case* saat memasukkan input stopwords yaitu kata-kata yang sering muncul seperti *the, a, at, in* dalam bahasa inggris, hasilnya adalah sebagai berikut

The screenshot shows the SABEB search interface. At the top, there are links for 'sabeb', 'Search', and 'Upload'. A green notification bar at the top left says 'Searching succeeded!'. Below this, there is a search section with a 'Search' label, a 'Query' input field containing the word 'the', and a 'Search' button. To the right of the search section is a 'Side List' titled 'Here is a list of documents used'. It contains a list of documents used, including 'Analysis At pivotal moments of 2020, Trump failed to win over doubters.txt', 'Biden urges healing and cooperation after defeating Trump for US presidency.txt', 'Canada's Bombardier faces UK bribery probe over plane sales.txt', 'Clashes erupt in protests against new Indonesian jobs law.txt', 'Consumer companies pitch discounts, bargain products as economic malaise looms.txt', and 'Exclusive Indonesia rejected US request to host spy planes - officials.txt'. Below the search section, there is a section showing 'Similarity : 0 %' and 'Total Words : 927'. The main content area displays the title 'Analysis At pivotal moments of 2020, Trump failed to win over doubters' and a snippet of text: 'WASHINGTON (Reuters) - Donald Trump had reason to count on the loyalty of the large chunk of Americans who drove his improbable election victory in 2016.'

The screenshot shows the SABEB search interface. At the top, there are links for 'sabeb', 'Search', and 'Upload'. Below this, there is a search section with a 'Search' label, a 'Query' input field containing the word 'the', and a 'Search' button. To the right of the search section is a 'Side List' titled 'Here is a list of documents used'. It contains a list of documents used, including 'Analysis At pivotal moments of 2020, Trump failed to win over doubters.txt', 'Biden urges healing and cooperation after defeating Trump for US presidency.txt', 'Canada's Bombardier faces UK bribery probe over plane sales.txt', 'Clashes erupt in protests against new Indonesian jobs law.txt', 'Consumer companies pitch discounts, bargain products as economic malaise looms.txt', and 'Exclusive Indonesia rejected US request to host spy planes - officials.txt'. Below the search section, there is a section showing 'Similarity : 0 %' and 'Total Words : 927'. The main content area displays the title 'Analysis At pivotal moments of 2020, Trump failed to win over doubters' and a snippet of text: 'WASHINGTON (Reuters) - Donald Trump had reason to count on the loyalty of the large chunk of Americans who drove his improbable election victory in 2016.'

Dapat dilihat bahwa stopwords tidak dapat dicari atau tidak dianggap sebagai input yang 'valid' sebagai query karena stopwords dihilangkan saat pemrosesan artikel.

Test-case selanjutnya adalah saat input yang didapat adalah kalimat/kata biasa, *test-case* ini dibagi dua antara kalimat/kata yang ada di salah satu dokumen dan yang tidak ada (bahasa Indonesia).

Pertama adalah *test-case* saat kalimat/kata query ada di dalam dokumen, hasilnya adalah sebagai berikut

sabeb Search Upload

Searching succeeded!

Term	Query	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
trump	1	21	25	0	0	0	0	10	9	0	0	0	0	0	0
biden	1	3	26	0	0	0	0	5	1	0	0	0	0	0	0

Search

Query

trump biden

Search

Side List

Here is a list of documents used

Documents used :

Analysis At pivotal moments of 2020, Trump failed to win over doubters.txt

Biden urges healing and cooperation after defeating Trump for US presidency.txt

Canada's Bombardier faces UK bribery probe over plane sales.txt

Clashes erupt in protests against new Indonesian jobs law.txt

Consumer companies pitch discounts, bargain products as economic malaise looms.txt

Exclusive Indonesia rejected US

Search

Query

trump biden

Search

Similarity : 50.0 %

Total Words : 1493

Biden urges healing and cooperation after

Sementara untuk kalimat/kata yang tidak ada pada dokumen manapun hasilnya adalah sebagai berikut

sabeb Search Upload

Searching succeeded!

Term	Query	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
aku	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
suka	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
kamu	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Search

Query

aku suka kamu

Search

Side List

Here is a list of documents used

Documents used :

Analysis At pivotal moments of 2020, Trump failed to win over doubters.txt

Biden urges healing and cooperation after defeating Trump for US presidency.txt

Canada's Bombardier faces UK bribery probe over plane sales.txt

Clashes erupt in protests against new Indonesian jobs law.txt

Consumer companies pitch discounts, bargain products as economic malaise looms.txt

Exclusive Indonesia rejected US request to host spy planes -

Search

Query

aku suka kamu

Search

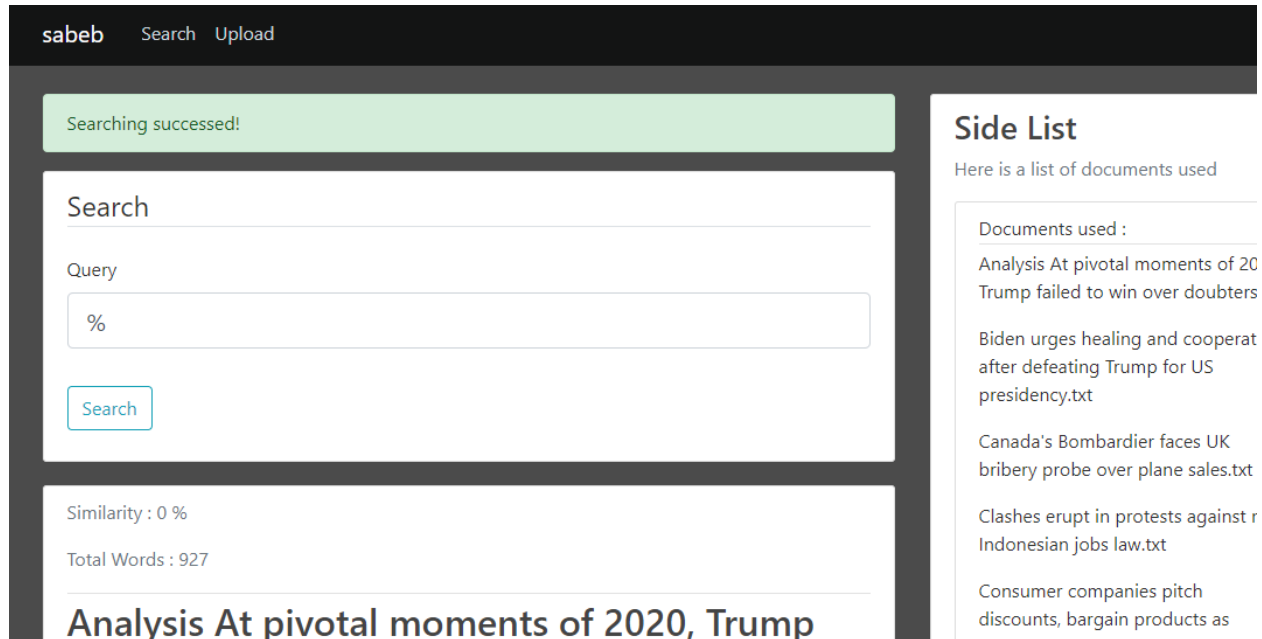
Similarity : 0.0 %

Total Words : 927

Analysis At pivotal moments of 2020, Trump

Dapat dilihat bahwa untuk kata-kata yang tidak ada atau tidak ditemukan pada dokumen manapun akan menghasilkan similarity 0 dan artikel-artikel yang ditampilkan tetap mengurut sesuai abjad.

Lalu ada juga *test-case* apabila input query adalah sebuah simbol



sabeb Search Upload

Searching succeeded!

Search

Query

%

Search

Similarity : 0 %

Total Words : 927

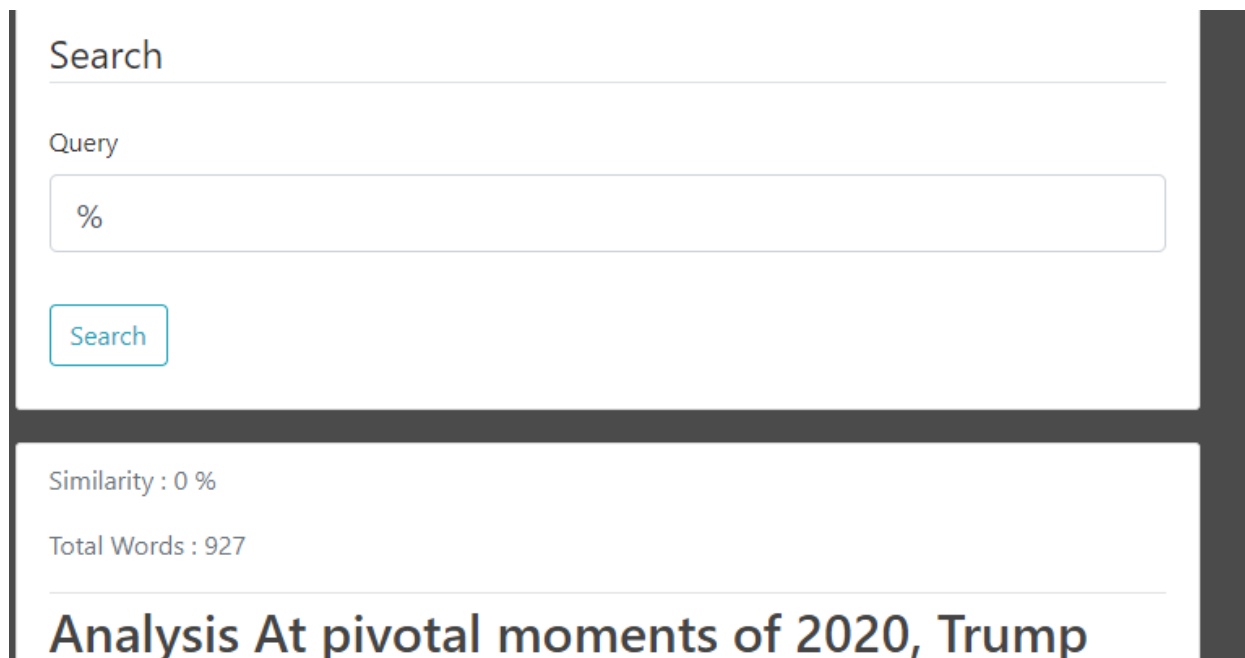
Analysis At pivotal moments of 2020, Trump

Side List

Here is a list of documents used

Documents used :

- Analysis At pivotal moments of 2020 Trump failed to win over doubters
- Biden urges healing and cooperat after defeating Trump for US presidency.txt
- Canada's Bombardier faces UK bribery probe over plane sales.txt
- Clashes erupt in protests against r Indonesian jobs law.txt
- Consumer companies pitch discounts, bargain products as



Search

Query

%

Search

Similarity : 0 %

Total Words : 927

Analysis At pivotal moments of 2020, Trump

Hasil input query dengan simbol '%' tidak didapatkan karena simbol '%' dianggap sebagai *special characters* dan juga dihilangkan dalam pemrosesan dokumen/artikel sehingga hasil search menampilkan seluruh dokumen menurut abjad dengan similaritas 0.

Dan *test-case* yang terakhir adalah input gabungan dari kalimat/kata biasa, angka, dan simbol sebagai querynya.

sabeb Search Upload

Searching succeeded!

Term	Query	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14
trump	1	21	25	0	0	0	0	10	9	0	0	0	0	0	0
26	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Search

Query

trump 26%

Search

Side List

Here is a list of documents used

Documents used :

Analysis At pivotal moments of 2020, Trump failed to win over doubters.txt

Biden urges healing and cooperation after defeating Trump for US presidency.txt

Canada's Bombardier faces UK bribery probe over plane sales.txt

Clashes erupt in protests against new Indonesian jobs law.txt

Consumer companies pitch discounts, bargain products as economic malaise looms.txt

Search

Query

trump 26%

Search

Similarity : 34.003917230155714 %

Total Words : 927

Analysis At pivotal moments of 2020, Trump

Meskipun terdapat sebuah simbol tetapi, kata dan angka tetap dianggap sebagai input query yang 'valid' dan dokumen yang bersesuaian akan ditampilkan.

BAB V

SIMPULAN, SARAN, DAN REFLEKSI

Kesimpulan

Kami berhasil membuat sebuah *search-engine* sederhana yang dapat melakukan proses pencarian sederhana dari file-file lokal yang ada. Untuk file-file lokal hasil pencarian, kami menggunakan metode *web scrap* dari dua buah page yang diambil dari internet dan artikel-artikelnya sebagai file lokal tersebut. Kami memanfaatkan Flask (Python) sebagai framework pembuatan website ini dan html css untuk tampilan website. Website yang berhasil kami buat pun dapat menampilkan similarity dari query dengan masing-masing dokumen dan menampilkan jumlah kemunculan sebuah query pada dokumen-dokumen yang ada dalam sebuah tabel. Kami juga berhasil membuat sebuah fitur *Upload* agar pengguna dapat mengunggah file yang diinginkan secara manual ke storage lokal pada website kami. Dari tugas besar ini, kami melihat bahwa model ruang vektor memiliki banyak sekali kegunaan dan aplikasinya terutama pada *information retrieval* dan pembuatan *search-engine* seperti ini.

Saran

Program yang kami kembangkan ini masih sangat jauh untuk dapat dibilang sempurna sehingga saran kami pada tugas besar kali ini adalah program ini mungkin dapat dikembangkan kembali pada masa yang mendatang. Program dapat dikembangkan dengan mengubah algoritma agar semakin praktis dan efektif dalam eksekusinya maupun mengadakan fitur-fitur tambahan seperti dapat menampilkan dan mengunggah gambar sebagai query atau hasil query. Pada bagian *front-end* pun dapat dikembangkan lebih jauh sehingga tampilan *search-engine* dapat lebih menarik dan praktis.

Refleksi

Dalam membuat tugas besar ini, kami bertemu dengan beberapa kendala. Meskipun begitu, terdapat sebuah kendala utama yang paling mendasar yaitu kurangnya pengalaman kami dalam *web development* sehingga kami masih bingung dalam menggunakan bahasa-bahasa yang dikhususkan untuk *web development* seperti html css termasuk frameworknya yaitu Flask dan juga cara menyambungkan antara *front-end* dan *back-end* dari website kami. Tetapi dalam mengerjakan tugas besar ini, kami sudah berusaha yang terbaik dan melalui kendala/hambatan tersebut sehingga kami pun yakin dengan selesainya tugas besar ini, segala sesuatu yang menjadi hambatan saat ini tidak akan menjadi hambatan lagi pada masa yang akan datang.

DAFTAR REFERENSI

eprints.umm.ac.id. *Information Retrieval*

[Diakses tanggal 8 November 2020]:

<http://eprints.umm.ac.id/37604/3/jiptummpp-gdl-fariskadwi-47791-3-bab2.pdf>

informatikalogi.com. *Vector Space Model (VSM) dan Pengukuran Jarak pada Information Retrieval (IR)*

[Diakses tanggal 8 November 2020]:

<https://informatikalogi.com/vector-space-model-pengukuran-jarak/>

Rinaldi Munir. *Aplikasi Dot Product pada Sistem Temu Balik*

[Diakses tanggal 8 November 2020]:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-12-Aplikasi-dot-product-pada-IR.pdf>

geeksforgeeks.org. *Removing Stop Words with NLTK Python*

[Diakses tanggal 9 November 2020]:

<https://www.geeksforgeeks.org/removing-stop-words-nltk-python/#:~:text=What%20are%20Stop%20words%3F,result%20of%20a%20search%20query>