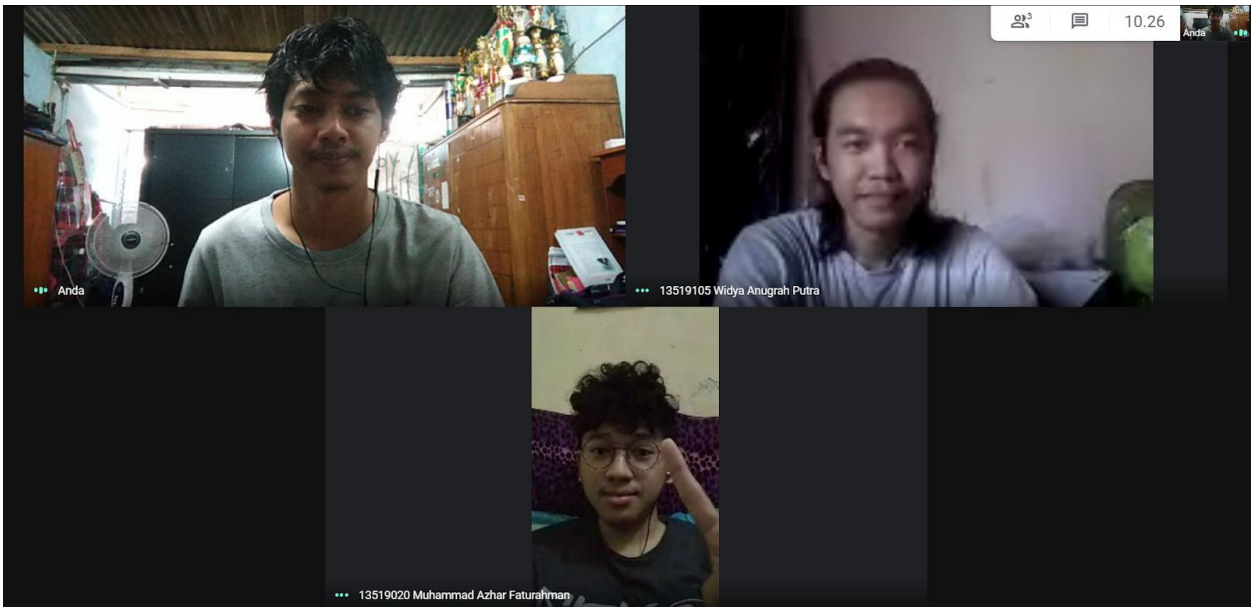
LAPORAN TUGAS BESAR II

IF2123 ALJABAR LINIER DAN GEOMETRI



Laporan ini dibuat untuk memenuhi tugas

Mata Kuliah IF 2123 Aljabar Linier dan Geometri

Disusun Oleh :

Kelompok 17

Muhammad Azhar Faturahman (13519020)

Widya Anugrah Putra (13519105)

Rezda Abdullah Fachrezzi (13519194)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER I TAHUN 2020/2021

DAFTAR ISI

BAB 1 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 2 TEORI SINGKAT

1. Temu Balik Informasi

   Temu balik informasi (*information retrieval* / IR) adalah proses menemukan kembali (*retrieval*) informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis. IR umumnya digunakan pada pencarian informasi yang isinya tidak terstruktur, seperti dokumen atau laman web.

   Aplikasi IR yang paling banyak digunakan adalah membuat Search Engine, seperti Google, Bing, Yahoo, dll.

2. *Information Retrieval* dengan Model Ruang Vektor

   Salah satu model IR adalah model ruang vektor, model ini menggunakan teori di dalam aljabar vektor. Misalkan terdapat n kata berbeda sebagai kamus kata (*vocabulary*) atau indeks kata (*term index*). Kata kata tersebut membentuk ruang vektor berdimensi n, setiap dokumen maupun query dinyatakan sebagai vektor $w = (w_1, w_2, ..., w_n)$ di dalam R^n . w_i menyatakan bobot setiap kata *i* di dalam query atau dokumen. Nilai w_i dapat juga menyatakan jumlah kemunculan kata tersebut dalam dokumen (*term frequency*).

3. *Cosine Similarity*

   Penentuan dokumen mana yang relevan dengan query dipandang sebagai pengukuran kesamaan (*similarity measure*) antara query dengan dokumen. Semakin sama suatu vektor dokumen dengan vektor query, semakin relevan dokumen tersebut dengan query. Kesamaan (sim) antara dua vektor $Q = (q_1, q_2, ..., q_n)$ dan $D = (d_1, d_2, ..., d_n)$ diukur dengan rumus *cosine similarity* yang merupakan bagian dari rumus perkalian titik (*dot product*) dua buah vektor:

$$Q \bullet D = ||Q|| \, ||D|| \, cos(\theta)$$

$$sim(Q, D) = cos(\theta) = \frac{Q \bullet D}{||Q|| \, ||D||}$$

dengan $Q \bullet D$ adalah perkalian titik yang didefinisikan sebagai :

$$Q \bullet D = q_1 d_1 + q_2 d_2 + ... + q_n d_n$$

Jika cos = 1, berarti = 0, vektor Q dan D berimpit , yang berarti dokumen D sesuai dengan query Q. Jadi, nilai cosinus yang besar mendekati 1) mengindikasikan bahwa dokumen cenderung sesuai dengan query. Setiap dokumen di dalam koleksi dokumen dihitung kesamaannya dengan query dengan rumus cosinus di atas. Selanjutnya hasil perhitungan di ranking berdasarkan nilai cosinus dari besar ke kecil sebagai proses pemilihan dokumen yang yang dekat ” dengan query. Perangkingan tersebut menyatakan dokumen yang paling relevan hingga yang kurang relevan dengan query. Nilai cosinus yang besar menyatakan dokumen yang relevan, nilai cosinus yang kecil menyatakan dokumen yang kurang relevan dengan query.

BAB 3 IMPLEMENTASI PROGRAM

PROGRAM PADA SISI PELADEN (server.py)

Sisi Peladen atau *Server Side* pada program ini menggunakan bahasa Python untuk menjalankan program, sisi peladen pada program ini digunakan untuk menciptakan sisi klien dan menjalankan algoritma-algoritma pencarian.

1. Inisialisasi *library*.

```
1   from flask import Flask, render_template, request, redirect
2   from backend import main
3   from time import time
```

Penjelasan:

Baris pertama mengambil fungsi Flask, render_template, request, dan redirect dari *library* flask. Flask digunakan untuk menginisialisasi sisi klien, render_template digunakan untuk me-*render* halaman dengan mengirim beberapa data yang akan diproses ke sisi klien, request digunakan untuk mengambil parameter yang dikirim melalui sisi klien, dan redirect digunakan untuk mengalihkan halaman pada sisi klien.

Baris kedua mengambil fungsi-fungsi pada program utama yang ada pada **main.py** dari folder backend, fungsi-fungsi tersebut akan digunakan untuk memproses pencarian.

Baris ketiga mengambil fungsi time dari *library* time, fungsi ini digunakan untuk mengambil waktu pada saat proses pencarian dimulai.

2. Inisialisasi sisi klien

```
5   # init
6   app = Flask(__name__, static_folder='frontend/static', template_folder='frontend/views')
7
```

Penjelasan:

Pada fungsi Flask tersebut, terdapat beberapa parameter. __name__ adalah variabel spesial yang sudah diimplementasikan oleh Python, berisi evaluasi dari program yang dijalankan, static_folder adalah parameter untuk akses fail-fail statis seperti logo atau lainnya untuk diakses dari sisi klien, template_folder adalah parameter untuk akses fail-fail yang akan ditampilkan pada sisi klien seperti halaman depan atau lainnya.

3. Router

Router atau pengarah pada sisi peladen digunakan untuk menampilkan atau mengalihkan sisi klien ke suatu halaman tertentu, pada program ini, fail-fail yang ditampilkan berada pada folder frontend yang sudah diinisiasikan pada Inisiasi sisi klien (2).

3.1. Halaman depan

```
9   @app.route('/')
10  def homepage():
11      """ halaman depan """
12      return render_template('index.html')
```

Penjelasan:

Bagian ini digunakan untuk mengalihkan sisi klien dengan alamat "/" ke fail **index.html**. Halaman ini menampilkan halaman depan pada *website*, digunakan untuk mengambil kata kunci dari pengguna.

3.2. Perihal

```
14  @app.route('/perihal')
15  def perihal():
16      """ halaman untuk perihal """
17      return render_template('perihal.html')
```

Penjelasan:

Bagian ini digunakan untuk mengalihkan sisi klien dengan alamat "/perihal" ke fail **perihal.html**. Halaman ini menampilkan cara kerja program, cara menggunakan program, dan nama-nama pembuat.

3.3. Search

```python
@app.route('/search', methods=['GET'])
def search():
  """
  halaman untuk mendapatkan hasil pencarian berdasarkan tiga paramater
  querysearch = kata kunci pencarian
  querytype = tipe pencarian (cepat atau akurat [lambat, dilakukan stemming])
  querydoc = jumlah dokumen yang akan dicari
  """
  doc = 160 # jumlah dokumen
  timebefore = time() # waktu (detik) sebelum dimulai pencarian, digunakan untuk menghitung lamanya pencarian

  # validasi request
  if not request.args.get('querydoc') or not request.args.get('querytype') or not request.args.get('querysearch'):
    return redirect("/")
  if not(int(request.args.get('querydoc')) > 0 and int(request.args.get('querydoc')) <= doc):
    return redirect("/")
  if int(request.args.get('querytype')) != 1 and int(request.args.get('querytype')) != 0:
    return redirect("/")

  # melewati validasi
  # mengembalikan data hasil pencarian dan parameter ke table.html untuk di-render
  a = main.main(request.args.get('querysearch'), int(request.args.get('querydoc')), int(request.args.get('querytype')))
  b = request.args.get('querysearch')
  c = request.args.get('querydoc')
  d = request.args.get('querytype')

  return render_template('table.html', data=a, len=[len(a[0]), len(a[1]), len(a[1][0])], query=b, doc=int(c), type=int(d), sec="%.2f"%(time()-timebefore))
```

Penjelasan:

Bagian ini digunakan untuk memproses pencarian yang dikirim dari halaman depan. Terdapat beberapa kamus yaitu doc sebagai total dari dokumen yang ada dan timebefore sebagai penanda waktu saat pencarian dimulai. Setelah kamus, terdapat validasi parameter yang dikirim dari halaman depan, parameter tersebut terdapat querysearch yang berisi kata kunci pencarian, querydoc yang berisi jumlah dokumen yang akan dicari, querytype yang berisi tipe pencarian (cepat atau akurat), jika validasi tidak terpenuhi, maka akan langsung dialihkan ke halaman depan tanpa memberi pesan eror. Namun, jika validasi terpenuhi, maka parameter-parameter tadi akan diproses dengan fungsi main pada program utama. Setelah itu, hasil dari proses pada main dan parameter-parameter tadi dikirim ke **table.html** untuk ditampilkan pada sisi klien.

3.4. Halaman dokumen

```python
@app.route('/doc/<path:path>')
def send_txt(path):
  '''
  Mengirim dokumen dengan judul path
  '''
  try:
    return render_template('doc.html', data=main.getSpecDoc(path, -1), judul=path)
  except FileNotFoundError:
    return redirect("/")
```

Penjelasan:

Bagian ini digunakan untuk mengalihkan alamat “/doc/[path]” ke **doc.html** dengan path adalah nama fail dokumen. Isi dari fail tersebut akan dikirim ke sisi klien. Jika fail tidak ditemukan, maka akan dialihkan ke halaman depan tanpa pesan eror.

3.5. *Error Handling* (404)

```
57  @app.errorhandler(404)
58  def invalid_route(e):
59      return redirect("/")
```

Penjelasan:

Mengatasi eror 404 (router tidak ditemukan) dengan mengalihkan sisi klien ke halaman utama.

PROGRAM UTAMA (main.py)

1. Import dan Inisialisasi library

```
import os.path
import re
import string
import pandas as pd
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory
from nltk.corpus import stopwords

# Module Initialization
factory = StemmerFactory()
stemmer = factory.create_stemmer()
```

Penjelasan :

Pada bagian ini program main.py akan mengimport library dan fungsi yang akan digunakan dalam program, library / fungsi yang digunakan antara lain library os, re, string, pandas, Sastrawi, dan nltk. Library os digunakan dalam mencari path atau alamat dari suatu dokumen di dalam folder, library re dan string digunakan dalam proses pembersihan dokumen, library pandas digunakan dalam pembentukan dataframe agar data lebih mudah untuk diolah, library Sastrawi digunakan dalam proses *stemming* kata, dan library nltk digunakan untuk membersihkan stopword pada dokumen.

2. Fungsi get_doc

```
def get_doc(N=15):
    """
    Fungsi ini digunakan untuk membaca file dari database sebanyak N dan disimpan ke documents \n
    Return document[0] : document, document[1] : namafile
    """
    # KAMUS LOKAL
    dir = "../test/"        # Untuk Server
    #dir = "../../test/"    # Untuk Testing

    list_File = os.listdir(dir)
    allFile = []    # Alamat dan nama file document

    for file_name in list_File:
        allFile.append(dir+file_name)

    documents = []    # List ini digunakan untuk menyimpan documents dari database
    i = 0
    # ALGORITMA
    for path in allFile:
        if (i<N):
            # Membaca file documents
            file = open(path,encoding='latin1')
            doc = file.read()
            # Menambah data documents dari file ke list
            filename = path.replace("../test/","")
            temp = [filename,doc]
            documents.append(temp)
            file.close()
            # Next instruction
            i += 1
        else:
            break

    return documents
```

Penjelasan :

Fungsi get_doc mengembalikan array yang berisi data-data dokumen yang ingin diambil. Fungsi ini menerima parameter N file yang ingin dibaca, jika N tidak ditentukan maka akan menggunakan N=15 sebagai defaultnya. Variabel dir berisi path folder dimana dokumen-dokumen disimpan, lalu array allFile berisi semua path dari masing-masing dokumen. Array documents akan berisi spesifikasi-spesifikasi dokumennya, dalam hal ini yang disimpan adalah nama file dan isi dokumennya. Selanjutnya dilakukan for loop untuk mengambil N buah dokumen dengan menggunakan fungsi open() untuk membuka dokumennya dan fungsi read() untuk memindahkan isi file ke variabel doc. Setelah itu variabel temp akan menerima nama filenya (berdasarkan path file lalu disesuaikan formatnya) dan isi dokumennya, lalu temp diappend ke array documents. Proses ini dilakukan sebanyak N kali. Fungsi ini mengembalikan array documents yang telah diproses.

3. Fungsi getSpecDoc

```
def getSpecDoc(docName,mode):
    ''' fungsi menerima nama dokumen tanpa ekstensi .txt dan mode penghapusannya, \n
    jika tidak ingin dihapus modenya -1, selain itu akan dihapus sesuai dengan paragraf cleaner  '''

    path = "../test/" + docName + ".txt"        # Untuk Server
    #path = "../../test/" + docName + ".txt"    # Untuk Testing
    file = open(path, encoding="latin1")
    doc = file.read()
    if (mode != -1):
        doc = text_cleaner(doc,mode)
    return doc
```

Penjelasan :

Fungsi getSpecDoc akan mengembalikan isi dokumen tertentu, fungsi ini menerima 2 parameter, yaitu nama file yang akan dibaca dan mode pembersihannya, jika modenya -1 maka tidak dilakukan pembersihan, selain itu akan dibersihkan sesuai dengan fungsi text_cleaner. Variabel path menyimpan path direktori dokumen-dokumen berada. Lalu variabel file membuka dokumen dan mengembalikan isinya.

4. Fungsi doc_cleaner

```
def doc_cleaner(documents,mode=0):
    """
    Membersihkan documents dan disimpan pada list clean_doc \n
    Terdapat 2 mode, 0 : Fast Cleansing, 1 : Accurate Cleansing
    Return documents yang sudah dibersihkan
    """

    # KAMUS LOKAL
    # pass_doc : setiap documents pada list documents
    clean_doc = [] # documents yang sedang dibersihkan

    for doc in documents:
        # Membersihkan tiap documents
        pass_doc = text_cleaner(doc[1],mode)
        # Menambah documents bersih
        clean_doc.append(pass_doc)

    return clean_doc
```

Penjelasan :

Fungsi doc_cleaner berguna untuk membersihkan dokumen-dokumen dari karakter-karakter yang tidak bermakna, lebih jelasnya akan dijelaskan di fungsi text_cleaner. Fungsi doc_cleaner menerima array yang berisi dokumen-dokumen serta mode pembersihan yang diperlukan. Dibentuk array baru bernama clean_doc yang akan berisi

dokumen-dokumen hasil proses pembersihan menggunakan fungsi text_cleaner. Setelah dilakukan iterasi tiap dokumen, dibersihkan dan diappend ke array clean_doc, fungsi akan mengembalikan array clean_doc .

5. Fungsi text_cleaner

```
def text_cleaner(text,mode=0,clear=True):
    """
    Membersihkan text dari karakter yang tidak diingikan
    """

    # KAMUS LOKAL
    clear_text = []

    # ALGORITMA
    # Membersihkan unicode ASCII yang tidak terpakai
    clear_text = re.sub(r'[^\x00-\x7F]+', ' ', text)
    # Membersihkan mention
    clear_text = re.sub(r'@\w+', '', clear_text)
    # Mode tidak terlalu membersihkan, tidak digunakan jika hanya ingin merapikan
    if clear:
        # Membuat semua huruf lower case
        clear_text = clear_text.lower()
        # Membersihkan punctuation
        clear_text = re.sub(r'[%s]' %re.escape(string.punctuation), ' ', clear_text)
        # Menghilangkan angka
        clear_text = re.sub(r'[0-9]', '', clear_text)
        # Membersihkan single alphabet
        clear_text = re.sub(r'\b[a-zA-Z]\b', '', clear_text)
    # Menghilangkan space berlebihan
    clear_text = re.sub(r'\s+', ' ', clear_text)

    # Mode stemming
    if (mode!=0):
        # Ini bagian yang tidak efisien dan menyebabkan program lambat
        clear_text = stemmer.stem(clear_text)

    return clear_text
```

Penjelasan :

Fungsi text_cleaner berguna untuk membersihkan karakter, karakter tidak penting, seperti unicode ASCII yang tidak terpakai, mention, space berlebihan. Jika diinginkan, fungsi ini juga bisa membuat tiap teksnya huruf kecil, membersihkan punctuation, menghilangkan angka, dan membersihkan single alphabet. Fungsi ini menerima parameter string, mode pembersihan dengan stemming, dan mode simplifikasi string yang defaultnya diatur ke True (jika True, fungsi akan melakukan apa yang tertulis di kalimat kedua paragraf ini). Mode pembersihan dengan stemming akan dilakukan jika modenya 1 (True) dan dilakukan dengan menggunakan library dari StemmerFactory. Setelah proses pembersihan selesai, karakter yang telah dibersihkan direturn.

6. Fungsi tf_docs

```
def tf_docs(clean_documents,query,mode):
    """
    Mengubah document bersih menjadi dataframe menggunakan pandas dan metode Term Frequency \n
    Return dataframe documents
    """
    # Inisialisasi dataframe
    df = pd.DataFrame([], columns=[0])

    # Kamus stopwords
    stop_words = set(stopwords.words('indonesian'))

    i = 0   # Dokumen pertama
    for doc in clean_documents:         # Iterasi tiap document
        df.loc[:,i] = 0                 # Inisialisasi nilai kolom dengan nol
        split_word = doc.split(' ')     # Split menjadi setiap kata

        # Menghilangkan Stopwords
        split_word = [w for w in split_word if not w in stop_words]

        # Vektorisasi setiap kata dalam dokumen
        for word in split_word:
            # Ini bagian yang tidak efisien dan menyebabkan program lambat
            if not(word in df.index):   # Apakah baris sudah ada?
                df.loc[word,:] = 0      # Jika belum, maka tambahkan baris baru dan inisialisasi semua dengan nol
                df.loc[word,i] = 1      # Lalu baris itu tambah 1
            else:
                df.loc[word,i] += 1     # Kalau udah ada tinggal increment

        i += 1      # Indeks dokumen
```

```
    """
    Menambah query menjadi vector dataframe
    """
    query_clean = text_cleaner(query,mode)
    split_word = query_clean.split(' ')

    # Menghilangkan Stopwords
    split_word = [w for w in split_word if not w in stop_words]

    df.loc[:,'query'] = 0   # Inisialisasi nilai kolom dengan nol

    # Vektorisasi setiap kata dalam query
    for word in split_word:
        if not(word in df.index):       # Apakah baris sudah ada?
            df.loc[word,:] = 0          # Jika belum, maka tambahkan baris baru dan inisialisasi semua dengan nol
            df.loc[word,'query'] = 1    # Lalu baris itu tambah 1
        else:
            df.loc[word,'query'] += 1   # Kalau udah ada tinggal increment

    # Merapikan data frame
    # df.sort_index(inplace=True)       # Sort Index, tidak terlalu perlu
    if ((df.index == '').any()):
        df = df.drop([''])              # Hapus Index kosong

    return df
```

Penjelasan :

Fungsi tf_docs berguna dalam membentuk dataframe dari setiap dokumen dan query, dataframe ini akan merepresentasi vektor setiap dokumen dan query. Setiap baris menyatakan term, dan setiap kolom akan menyatakan dokumen dan query. Elemen pada dataframe ini menyatakan banyaknya term pada suatu dokumen atau query.

Contoh bentuk dataframe :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	query
pemilu	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0
presiden	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	9.0	0.0	0
amerika	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0	1
serikat	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0
pilpres	6.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	19.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
berkaitan	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0
fleksibilitas	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0
kepatuhan	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0
protokol	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0
tutup	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0

7. Fungsi cos_similiarity

```python
def cos_similiarity(df):
    '''
    Fungsi ini digunakan untuk menghitung cosine similiarity dari dataframe \n
    dokumen dan query yang sudah dibersihkan
    '''
    # cos_sim = []      # Indeks menyatakan urutan dokumen
    norm_doc = []       # Indeks menyatakan urutan dokumen
    norm_query = 0      # Inisialisasi
    dot_doc = []        # Indeks menyatakan urutan dokumen

    # Pembentukan vektor disesuaikan dengan kamus kata
    col_df = df.columns     # List nama kolom
    row_df = df.index       # List nama index

    # Normalisasi Vektor dari Query
    sum = 0
    for row in row_df:
        sum += (df.loc[row,'query'])**2
    norm_query = sum**(1/2)

    # Normalisasi Vektor dan menghitung Dot Product dari Dokumen
    for col in col_df:
        if (col != 'query'):    # Biar cuma dokumen doang
            sum_norm = 0
            sum_dot = 0
            for row in row_df:
                # Normalisasi Vektor
                sum_norm += (df.loc[row,col])**2
                # Dot Product
                sum_dot += df.loc[row,col] * df.loc[row,'query']

            norm_doc.append(sum_norm**(1/2))    # Jangan lupa diakarkan
            dot_doc.append(sum_dot)

    # Inisialisasi nilai cos_sim
    cos_sim = [0 for i in range(len(col_df)-1)]
    # Hitung nilai cos_sim
    for i in range(len(col_df)-1):  # Kolom query tidak termasuk
        if ((norm_doc[i])!=0):      # Hati-hati ada norma vektor yang nol
            cos_sim[i] = dot_doc[i]/(norm_query*norm_doc[i])

    return cos_sim
```

Penjelasan :

Fungsi ini berguna untuk menghitung cosine similarity antara dokumen-dokumen dengan query. Fungsi ini menerima dataframe yang berisi banyaknya term tiap-tiap dokumen dan query. Pertama, fungsi ini akan menghitung norm dari vektor query, lalu karena dataframe yang dipakai bersifat columnwise, kita mengiterasi tiap-tiap kolom (dokumen) lalu dilanjutkan mengiterasi tiap-tiap baris (term) lalu dihitung norm vektor dokumen beserta hasil perkalian dot dari vektor dokumen dengan vektor query. Setelah itu fungsi membentuk array cos_sim yang akan menyimpan hasil cosine similarity, dengan membagi perkalian dot vektor dengan norma-norma vektornya. Fungsi setelah itu mengembalikan array cos_sim yang berisi *cosine similarity* dari setiap dokumen.

8. Fungsi dataToList(df)

```
def dataToList(df):
    '''
    Mengubah dataframe menjadi list
    '''
    #list_data = []

    # Hapus term yang tidak diquery
    idx = (df['query'] != 0)   # Cari kata yang tidak nol di kolom query
    df_new = df.loc[idx,:]     # Bentuk dataframe baru

    # Ubah urutan kolom query pada dataframe lama lalu simpan di dataframe baru
    col = df_new.columns.tolist()
    col = col[-1:] + col[:-1]
    df_new = df_new[col].astype(int)    # Ubah tipe data jadi integer

    # Mengubah nama kolom
    new_name = ['query']
    for i in range(len(col) - 1):        # Kurangi 1 karena ada query
        new_name.append(i+1)                # Pake urutan nama file
    df_new.columns = new_name

    # Jadikan list
    col_name = [['term']+df_new.columns.tolist()]
    list_data = col_name + df_new.reset_index().values.tolist()     #Menambah setiap baris

    return list_data
```

Penjelasan :

Pada fungsi ini, dataframe yang terdiri atas vektor *term frequency* dokumen dan query akan diubah menjadi bentuk list, karena dataframe/tabel tersebut harus dikirim ke client melalui server, sedangkan server belum tentu akan support format dataframe, sehingga perlu diubah menjadi list. Selain itu list/tabel yang dikirim hanya akan terdiri atas term yang di-query saja untuk meringkas waktu pengiriman data dari server ke client dan meringkas tampilan Tabel Term pada frontend.

9. Fungsi fsDocs

```
def fsDocs(documents):
    fsd = []
    for docs in range(len(documents)):
        s = documents[docs][1]
        # hapus dulu unicode biar ganteng, kadang di kalimat pertama udah ada unicodenya
        clear = re.sub(r'[^\x00-\x7F]+', ' ', s)
        # kalimat pertama diakhiri tanda titik "." dan spasi selanjutnya,
        # kalau cuma titik nanti bisa berhenti di KOMPAS.com
        idx = clear.find('. ')
        temp = clear[0:idx]
        # tambahin titik yang ikutan kehapus
        temp = temp + '.'
        fsd.append(temp)
    return fsd
```

Penjelasan :

Fungsi fsDocs berguna untuk mencari kalimat pertama dari string. Fungsi ini menerima array yang elemennya berisi string, tiap stringnya akan dibersihkan dari unicode ASCII yang tidak diperlukan, setelah itu akan dicari index substring ". " dari string tersebut. ". " menandakan substring sebelumnya merupakan kalimat pertama. Lalu ambil subsring sebelum index yang sudah ditemukan, simpan substring tersebut ke variabel temp, lalu tambahkan tanda titik agar menjadi kalimat yang utuh seperti sedia kala. Hasil dari proses ini diappend ke array baru bernama fsd, setelah dilakukan untuk tiap dokumen, fungsi ini mereturn array fsd.

10. Fungsi sumWord

```
def sumWord(clean_doc):
    sumW = []
    for docs in range(len(clean_doc)):
        s = clean_doc[docs]
        temp = len(s.split())
        sumW.append(temp)
    return sumW
```

Penjelasan :

Fungsi ini berguna untuk menghitung banyaknya kata pada setiap dokumen yang sudah dibersihkan untuk nanti dikirim kepada client.

11. Fungsi main

```
def main(query="master wiwid panutan kita",N=15,mode=0):
    '''
    PROGRAM UTAMA
    query = query document yang paling sesuai \n
    N = banyaknya document \n
    mode = 0 (standart, fast, default), 1 (dilakukan stemming) \n
    return : list document dan cos_sim, dan list term yang di-query
    '''
    # Inisialisasi
    documents = []          #document[i][0] : nama_file, document[i][1] : text dokumen,
    clean_docs = []         #document[i][2] : kalimat pertama, document[i][3] : banyak kata
    list_term = []

    # Dapatkan dokumen dan bersihkan
    documents = get_doc(N)
    firstSentence = fsDocs(documents)           # simpan kalimat pertama
    clean_docs = doc_cleaner(documents,mode)    # bersihkan dokumen
    wordSum = sumWord(clean_docs)               # hitung banyak kata tiap dokumen
    # Buat dataframe dengan pandas
    df = tf_docs(clean_docs,query,mode)         # note : Urutan dataframe sesuai dengan urutan dokumen
    # Hitung cosine simiarity dari tiap dokumen
    sim = cos_similiarity(df)
    # List term yang diquery
    list_term = dataToList(df,documents)

    # Gabungkan cosine similiarity dan document kedalam satu array
    sim_doc = []
    for i in range(len(documents)):
        documents[i][1] = text_cleaner(documents[i][1],0,False)  # Bersihkan sedikit dokuments
        temp = [sim[i],documents[i][0],documents[i][1],firstSentence[i],wordSum[i]]
        sim_doc.append(temp)

    # Urutkan berdasarkan cosine similiarity
    sim_doc.sort(reverse=True)

    return sim_doc, list_term
```

Penjelasan:

Fungsi main akan mengambil parameter query, banyak dokumen, serta mode stemming. Lalu dilakukan proses penghitungan cosine similarity, pencarian first sentence, dan penghitungan jumlah kata. Selain itu fungsi ini juga membentuk list term berdasarkan term query. Proses pada fungsi main dilakukan menggunakan fungsi-fungsi di atas.

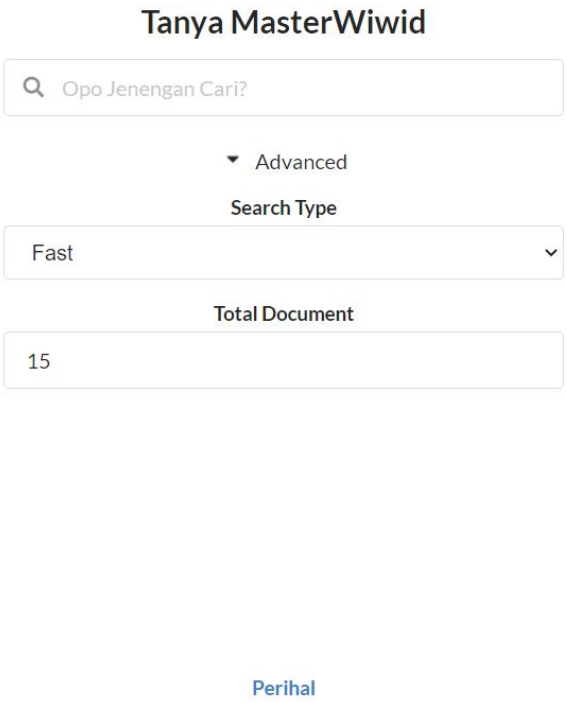
Sisi Klien

Sisi klien adalah tempat di mana pengguna menggunakan program ini. Sisi klien pada program ini menggunakan *template* dari Semantic UI.

1. index.html

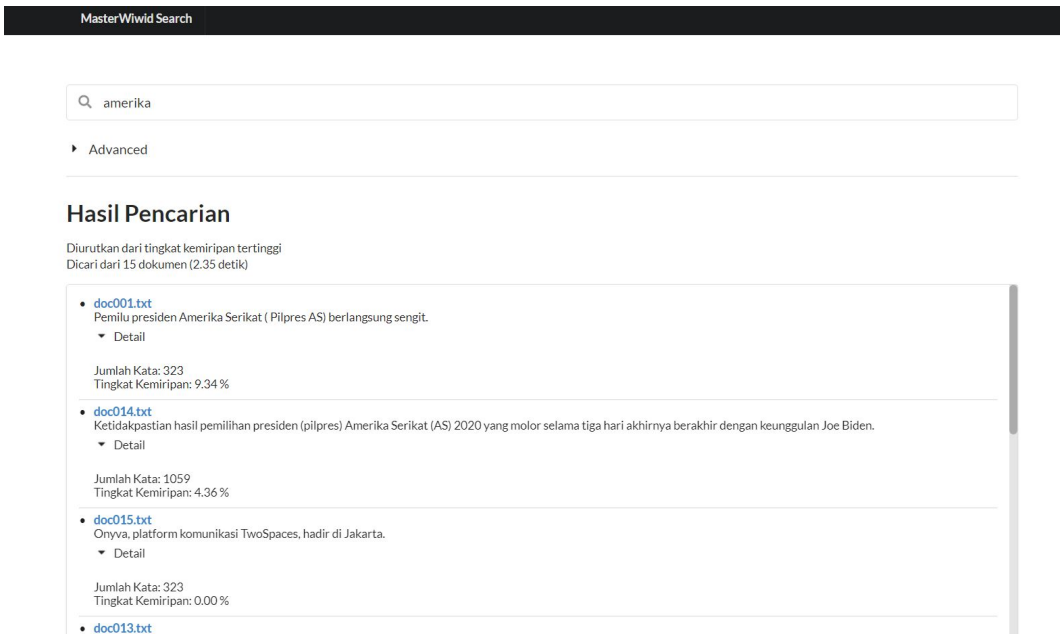
Fail ini digunakan sebagai halaman depan program. Terdapat beberapa kotak *input*, yaitu kotak untuk mengisi kata kunci pencarian, tombol *advanced* untuk menampilkan kotak

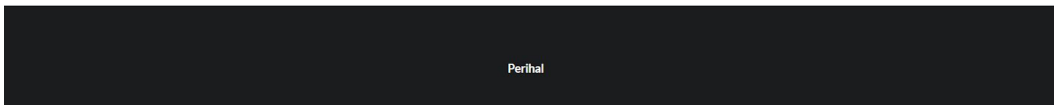
input untuk pencarian yang lebih lanjut, terdapat kotak untuk memilih tipe pencarian dan banyaknya dokumen. Di bagian bawah juga terdapat *hyperlink* untuk ke halaman perihal.



2. table.html

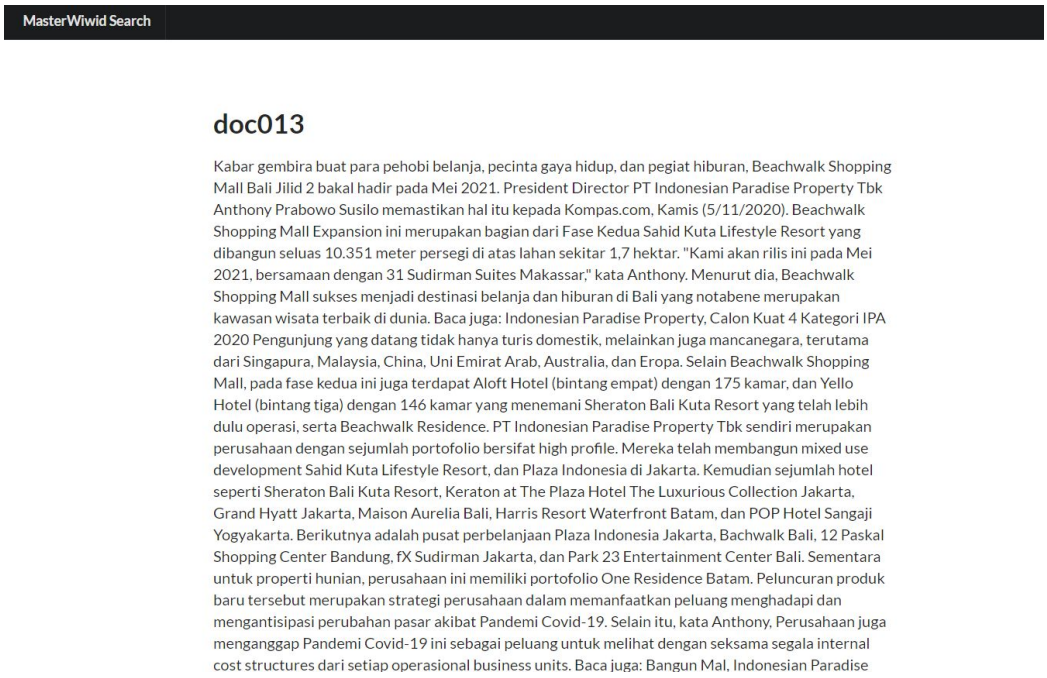
   Fail ini digunakan untuk menampilkan hasil pencarian beserta tabel *term*.





3. doc.html

   Fail ini digunakan untuk menampilkan isi dokumen dengan judul tertentu.



4. perihal.html

   Fail ini digunakan untuk menampilkan perihal.

MasterWiwid Search

Penjelasan singkat:

Mesin Pencari sederhana ini menggunakan *Cosine Similarity* sebagai algoritma pencariannya. Untuk *Backend*, program ini menggunakan *framework* **Flask** dari **Python**. Untuk *Frontend*, program ini menggunakan *template* dari Semantic UI. Dokumen yang diujikan pada program ini terdapat 160 dokumen yang semuanya diambil dari KOMPAS dengan metode *Web Scraping*.

Cara Pakai:

1. Isi kotak pencarian dengan kata kunci, lalu tekan ***enter***.
2. Untuk pencarian tingkat lanjut, terdapat dua kotak yaitu tipe pencarian dan banyaknya dokumen untuk dicari, ini dibebaskan kepada Anda.

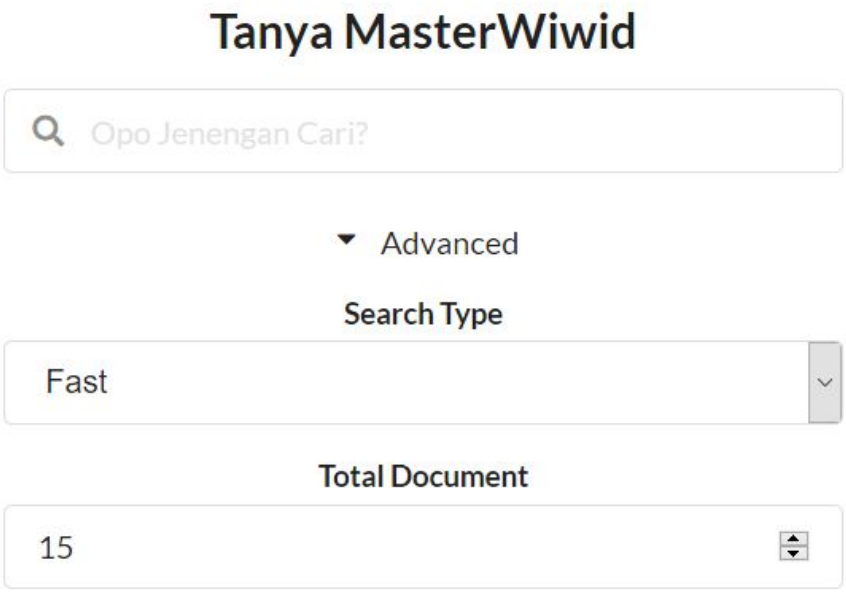
Tipe pencarian tingkat lanjut terdapat pilihan cepat dan akurat, tipe pencarian cepat tidak dilakukan stemming, tipe pencarian akurat dilakukan stemming dan cenderung lambat

Pembuat:

1. Muhammad Azhar Faturahman (K-4) 13519020
2. Widya Anugrah Putra (K-1) 13519105
3. Rezda Abdullah Fachrezzi (K-2) 13519194

BAB 4 EKSPERIMEN

Tampilan depan Search Engine

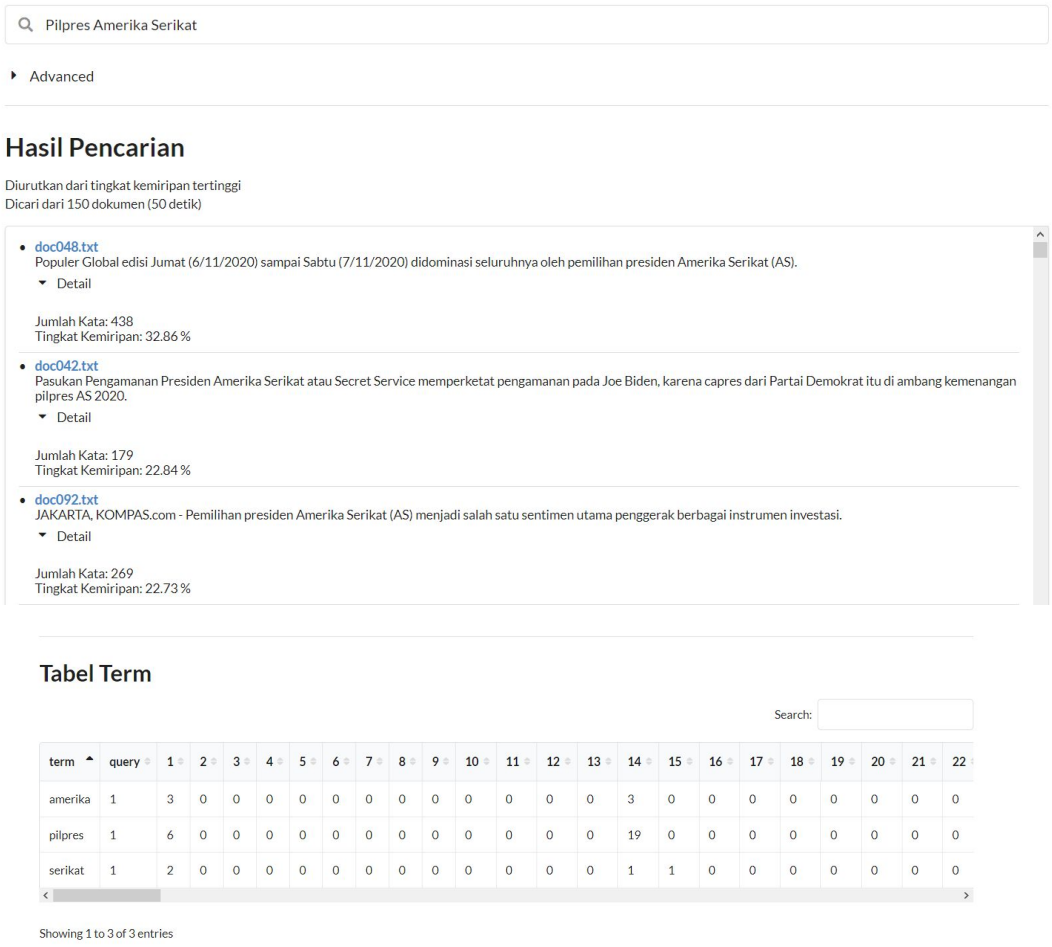


A. Pengujian

1. Kasus A

Mencari dokumen tentang “Pilpres Amerika Serikat” tanpa *stemming* dari database sebanyak 150 buah dokumen

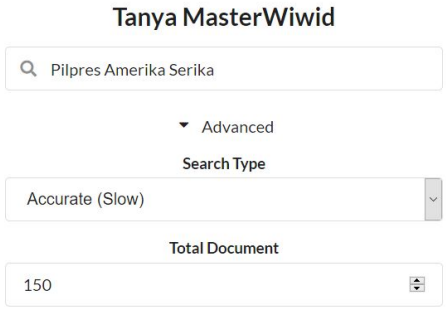
Hasil :



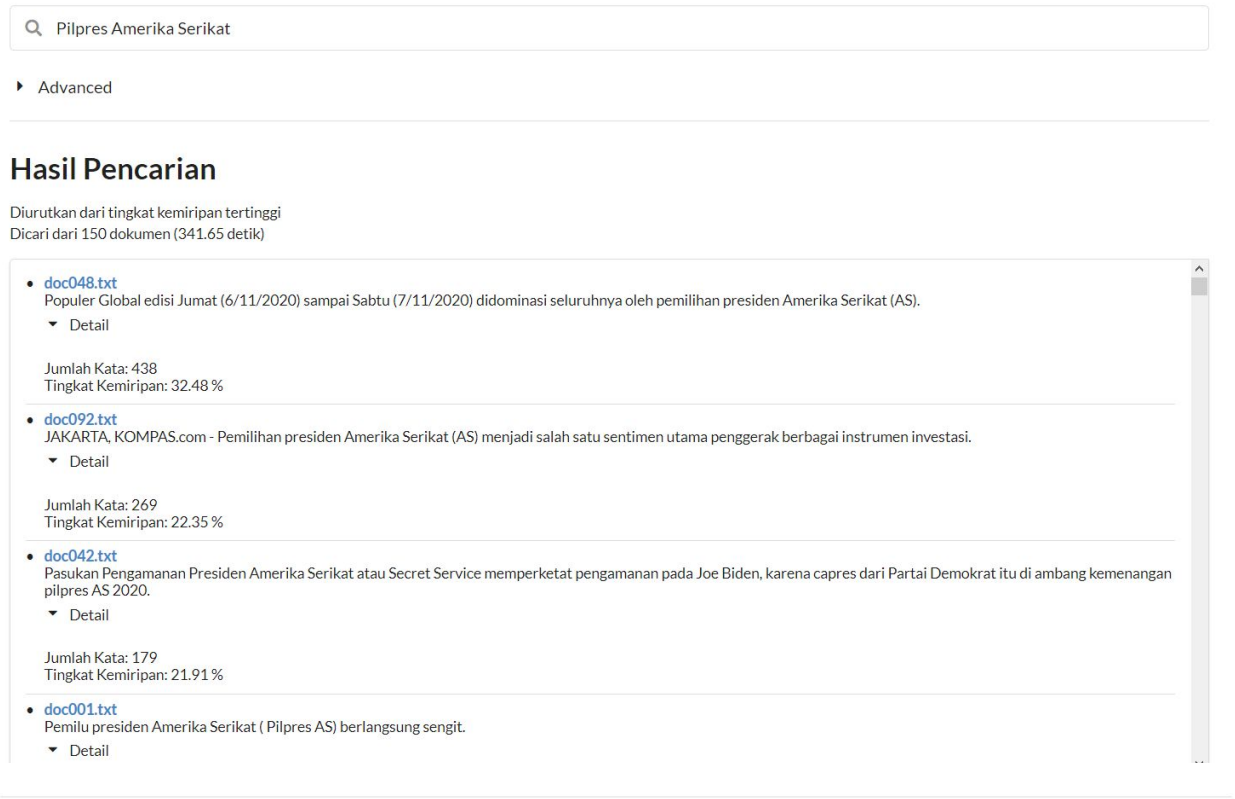
Dari hasil pencarian berhasil didapatkan dokumen yang sesuai dengan query, akan tetapi karena dokumen sangat banyak, diperlukan waktu pencarian selama 50 detik.

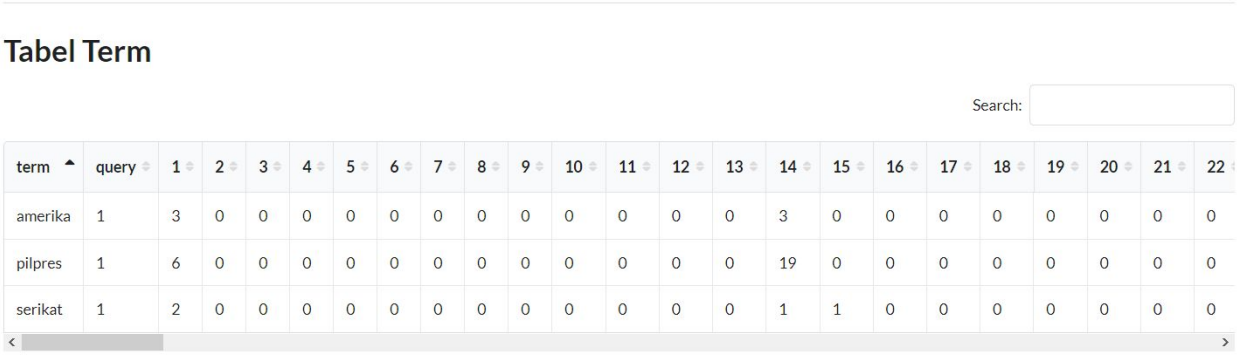
2. Kasus B

Mencari dokumen tentang “Pilpres Amerika Serikat” dengan *stemming* dari database sebanyak 150 buah dokumen



Hasil :





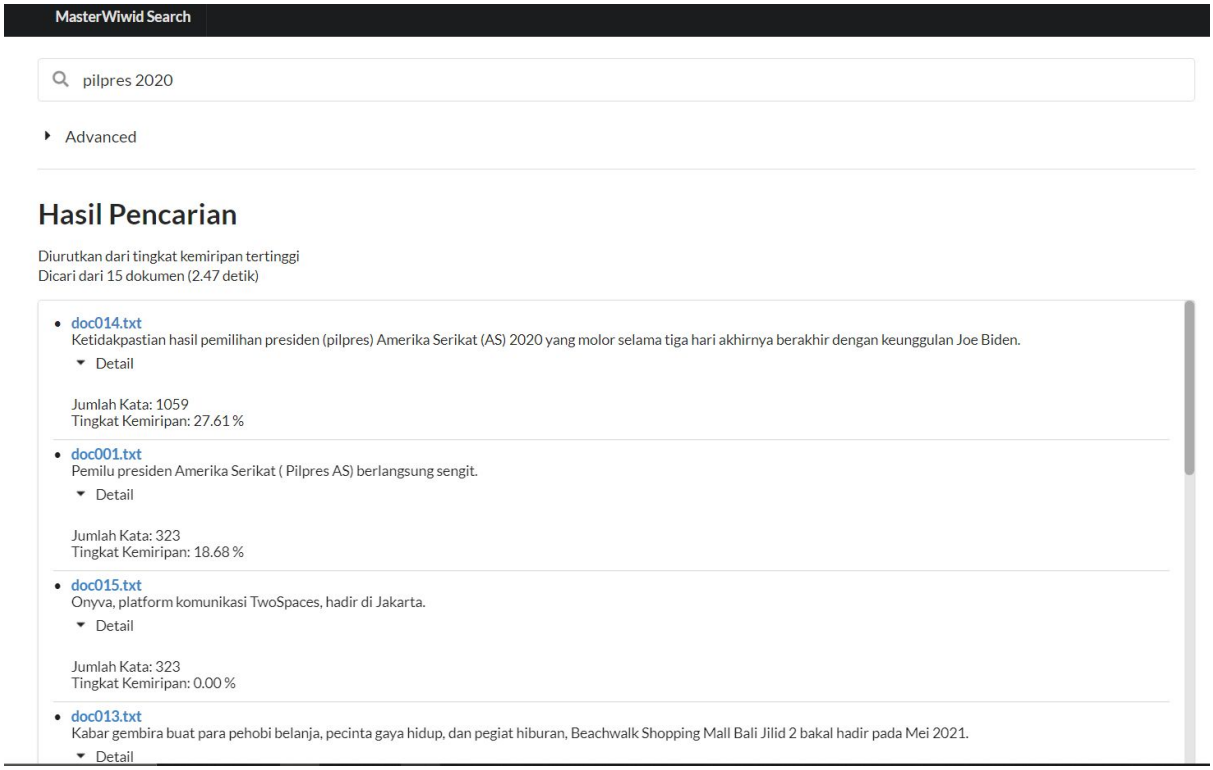
term	query	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
amerika	1	3	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0
pilpres	1	6	0	0	0	0	0	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0
serikat	1	2	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Didapatkan hasil yang tidak berbeda jauh dengan kasus A, namun diperlukan waktu pencarian yang lebih lama yaitu 340 detik.

3. Kasus C

Mencari dokumen dengan menambahkan angka sebagai kata kunci.

Hasil:



Tabel Term

Search:

term	query	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
pilpres	1	6	0	0	0	0	0	0	0	0	0	0	0	0	19	0

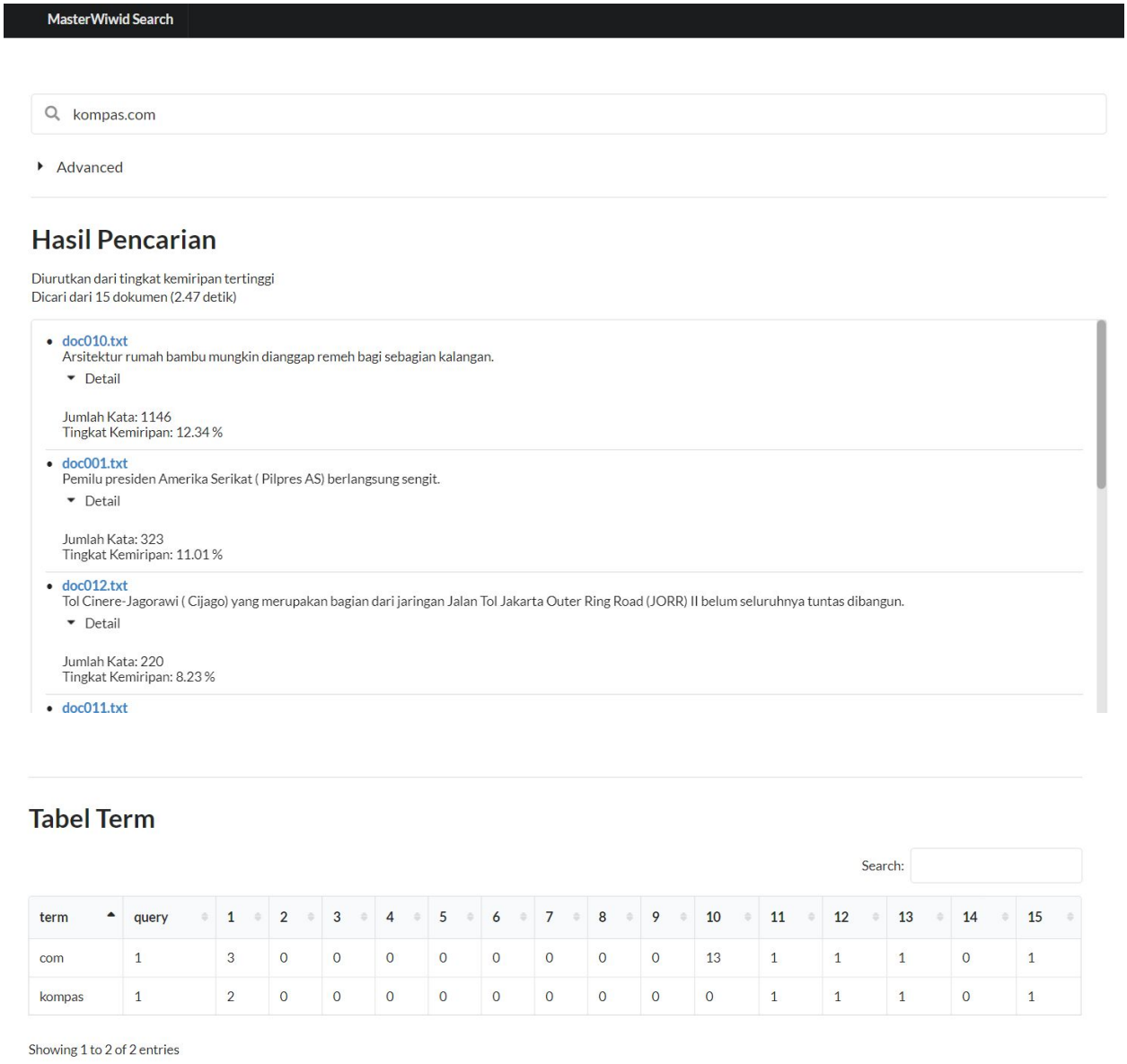
Showing 1 to 1 of 1 entries

Angka tersebut tidak muncul pada tabel *term* karena sudah terfilter oleh algoritma.

4. Kasus D

Menambahkan simbol pada kata kunci pencarian

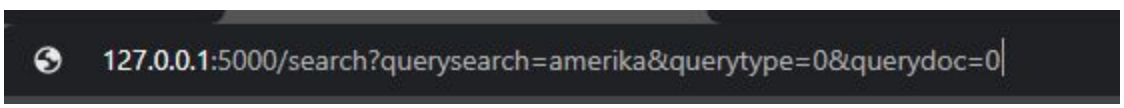
Hasil:



Simbol tersebut tidak muncul pada tabel *term* karena sudah terfilter oleh algoritma dan kata-kata yang dipisah simbol tersebut dijadikan *term* terpisah.

5. Kasus E

Secara gamblang membuka alamat situs tempat memproses pencarian tanpa melakukan pencarian melalui halaman depan dengan parameter yang salah.



Pada alamat tersebut, terdapat parameter yang tidak sesuai yaitu **querydoc** yang sama dengan nol, hal ini menyebabkan parameter tersebut tidak lolos validasi pada sisi peladen sehingga langsung dialihkan ke halaman depan. Hal ini berlaku juga untuk parameter lainnya; querytype haruslah nol atau satu; querysearch haruslah memiliki isi.

B. Analisis

1. Waktu Pencarian

   Waktu pencarian dokumen disertakan pada laman hasil pencarian. Untuk pencarian dengan total 15 dokumen atau lebih, pencarian memakan waktu lebih dari satu detik, hal ini disebabkan oleh *library* yang membersihkan *query* dan dokumen. Pada tipe pencarian akurat, dilakukan *stemming* terlebih dahulu dengan *library* Sastrawi, namun karena *library* Sastrawi yang tidak efisien, pencarian menjadi sangat lambat. Sebagai perbandingan, dengan kata kunci "Amerika", total 15 dokumen, dan pada pencarian pertama kali, metode pencarian cepat memiliki waktu pencarian sebesar 2,56 detik, sedangkan metode pencarian akurat atau dengan *stemming* memiliki waktu pencarian sebesar 83,92 detik. Penyebab lain lamanya pencarian adalah kemampuan komputasi dari gawai atau perangkat sisi peladen.
2. Analisis Kasus

   Sudah dicantumkan pada masing-masing kasus pada bagian **pengujian (A)**.

BAB 5 KESIMPULAN, SARAN, DAN REFLEKSI

Kesimpulan :

Program mesin pencari sederhana ini berhasil dibuat dan dijalankan sesuai ekspektasi. Program dapat mencari dokumen dengan tiga parameter, yaitu querysearch adalah kata kunci pencarian, querydoc adalah jumlah dokumen, querytype adalah tipe pencarian (cepat atau akurat). Akan tetapi efisiensi dari program mesin pencari yang kami buat ini masih tergolong tidak efisien karena memerlukan waktu yang sangat lama untuk melakukan vektorisasi kamus kata dokumen dan query, sehingga

melambatkan laju berjalannya program. Sehingga untuk melakukan pencarian masih memerlukan waktu yang tergolong lama.

Saran :

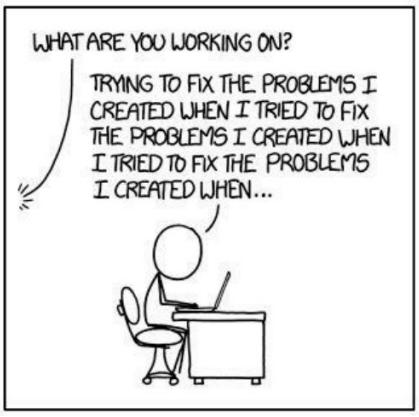
Untuk sisi peladen, didapat pencarian dokumen memerlukan waktu yang tidak sebentar, untuk hal ini, lebih baik digunakan *library* yang lebih efisien daripada Sastrawi. Selain itu, kecepatan pencarian juga dapat ditingkatkan dengan meningkatkan perangkat pada sisi peladen. Selain masalah waktu, sisi peladen juga dapat ditingkatkan keamanannya seperti menambah *error handler* dan menambah validasi-validasi.

Untuk sisi klien, laman-laman yang digunakan dapat ditingkatkan lagi *User Experience* dan *User Interface* nya.

Refleksi :

Dalam mengerjakan tugas besar ini, terdapat beberapa kendala. Diantaranya adalah kurangnya pengalaman kami dalam membangun website, kemudian program yang kami buat masih bergantung pada library yang tersedia pada bahasa pemograman Python.

Kesalahan tersebut murni datang dari kekurangan dan kemampuan kami, oleh karena itu kami akan berusaha hingga yang menjadi hambatan di saat ini tidak lagi menjadi hambatan di masa yang akan datang.



BAB 6 PEMBAGIAN TUGAS

NO	BAGIAN	PJ	STATUS
1	ALGORITMA SEARCH ENGINE	2 orang (Azhar, Wiwid)	Finish
2	BACKEND & API	1 orang (Rezda)	Finish
3	DOKUMEN	1 orang (Azhar)	Finish
4	FRONTEND WEB	2 orang (Wiwid, Rezda)	Finish

Note :

Backend menggunakan Python & Flask

Frontend menggunakan *template* dari Semantic UI.

REFERENSI

“Aplikasi Dot Product pada sistem temu balik aplikasi” by Rinaldi Munir
https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2020-2021/Algeo-12-Aplikasi-dot-product-pada-IR.pdf

"Create A Simple Search Engine Using Python" by Irfan Alghani Khalid
https://link.medium.com/yEtxO932Kab

"Implementing the TF-IDF Search Engine" by Kartheek Akella
https://link.medium.com/UcdqCt92Kab

“Create APIs with python and flask” by programming historian
https://programminghistorian.org/en/lessons/creating-apis-with-python-and-flask

“How to create a react flask project” by Miguel Grinberg
https://blog.miguelgrinberg.com/post/how-to-create-a-react--flask-project

“Removing Stop Words with NLTK Python”

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

"Sastrawi Library"
https://pypi.org/project/Sastrawi/

"Pandas Library"
https://pandas.pydata.org/

"The Most Loveable Website in The World"
https://stackoverflow.com/

"Semantic UI"
https://semantic-ui.com/

"Menjadi Developer Web dengan Python dan Flask Bagian I: Hello World"
https://www.codepolitan.com/menjadi-developer-web-dengan-python-dan-flask-hello-world-5a3b1af29b052