# Laporan Tugas Besar II

# IF2123 Aljabar Linier dan Geometri
# Aplikasi Dot Product pada Sistem Temu-balik Informasi
# Semester I Tahun 2020/2021


Dibuat oleh

James Chandra - 13519078
Hizkia Raditya Pratama Roosadi - 13519087
Nathaniel Jason - 13519108



Foto Bersama Versi Online


**Program Studi Teknik Informatika**
**Sekolah Teknik Elektro dan Informatika**
**Institut Teknologi Bandung**

**Bab 1**

**Abstraksi**

Hampir semua dari kita pernah menggunakan search engine, seperti google, bing dan yahoo! search. Setiap hari, bahkan untuk sesuatu yang sederhana kita menggunakan mesin pencarian Tapi, pernahkah kalian membayangkan bagaimana cara search engine tersebut mendapatkan semua dokumen kita berdasarkan apa yang ingin kita cari? Sebagaimana yang telah diajarkan di dalam kuliah pada materi vector di ruang Euclidean, temu-balik informasi (information retrieval) merupakan proses menemukan kembali (retrieval) informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis. Biasanya, sistem temu balik informasi ini digunakan untuk mencari informasi pada informasi yang tidak terstruktur, seperti laman web atau dokumen. Ide utama dari sistem temu balik informasi adalah mengubah search query menjadi ruang vektor Setiap dokumen maupun query dinyatakan sebagai vektor w = (w1, w2,..., wn) di dalam Rn , dimana nilai wi dapat menyatakan jumlah kemunculan kata tersebut dalam dokumen (term frequency). Penentuan dokumen mana yang relevan dengan search 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 tersebut dapat diukur dengan cosine similarity. Pada kesempatan ini, kalian ditantang untuk membuat sebuah search engine sederhana dengan model ruang vector dan memanfaatkan cosine similarity.

**Penggunaan Program**

Berikut ini adalah input yang akan dimasukkan pengguna untuk eksekusi program.

1. Search query, berisi kumpulan kata yang akan digunakan untuk melakukan pencarian
2. Kumpulan dokumen, dilakukan dengan cara mengunggah multiple file ke dalam web browser.

Perihal: link ke halaman tentang program dan pembuatnya (Konsep singkat search engine yang dibuat, How to Use, About Us). 3 Catatan: Teks yang diberikan warna biru merupakan hyperlink yang akan mengalihkan halaman ke halaman yang ingin dilihat.

Apabila menekan hyperlink , maka akan diarahkan pada sebuah halaman yang berisi full-text terkait dokumen 1 tersebut (seperti Search Engine). Anda dapat menambahkan menu lainnya, gambar, logo, dan sebagainya. Tampilan Front End dari website dibuat semenarik mungkin selama mencakup seluruh informasi pada layout yang diberikan di atas. Data uji berupa dokumen-dokumen yang akan diunggah ke dalam web browser. Format dan extension dokumen dibebaskan selama bisa dibaca oleh web browser (misalnya adalah dokumen dalam bentuk file txt atau file html). Minimal terdapat 15 dokumen berbeda. Tabel term dan banyak kemunculan term dalam setiap dokumen akan ditampilkan pada web browser dengan layout sebagai berikut.

| Term | Query | D1 | D2 | ... | D3 |
|---|---|---|---|---|---|
| Term1 | | | | | |
| Term2 | | | | | |
| ... | | | | | |
| TermN | | | | | |

Term Query D1 D2 ... D3 Term1 Term2 ... TermN Untuk menyederhanakan pembuatan search engine, terdapat hal-hal yang perlu diperhatikan dalam eksekusi program ini.

1. Silahkan lakukan stemming dan penghapusan stopwords pada setiap dokumen
2. Tidak perlu dibedakan antara huruf-huruf besar dan huruf-huruf kecil.
3. Stemming dan penghapusan stopword dilakukan saat penyusunan vektor, sehingga halaman yang berisi full-text terkait dokumen tetap seperti semula.
4. Penghapusan karakter-karakter yang tidak perlu untuk ditampilkan (jika menggunakan web scraping atau format dokumen berupa html)
5. Bahasa yang digunakan dalam dokumen adalah bahasa Inggris atau bahasa Indonesia (pilih salah satu) Petunjuk: silahkan gunakan library sastrawi atau nltk untuk stemming kata dan penghapusan stopwords

**Saran Pengerjaan**

Anda disarankan untuk membuat program testing pada backend terlebih dahulu untuk menguji keberhasilan dari perhitungan cosine similarity tersebut.

**Spesifikasi Tugas**

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**

## Retrieval Information

*Retrieval information* atau biasa disebut "sistem temu balik informasi" adalah suatu metode yang digunakan untuk menemukan kembali informasi - informasi yang relevan terhadap kebutuhan pengguna dari suatu kumpulan informasi secara otomatis. Salah satu aplikasi dari sistem temu balik informasi sangat dekat dengan kehidupan sehari - hari kita, yaitu *search engine* atau mesin pencarian yang terdapat pada jaringan internet, pengguna dapat mencari halaman web yang dibutuhkan dengan memasukan input berupa *search query* pada *search engine*.

Salah satu model sistem temu balik informasi adalah model ruang vektor. Model ini menggunakan teori yang diterapkan pada aljabar vektor. Misalkan terdapat *n* kata berada sebagai kamus kata atau indeks kata. Kata - kata tersebut membentuk suatu ruang vektor berdimensi *n*. Setiap dokumen maupun *query* dinyatakan sebagai vektor w = (w*1,* w*2, ...,* w*n*) di dalam R*n*, w*i* adalah bobot setiap kata *i* di dalam *query* atau dokumen. Nilai w*i* menyatakan frekuensi kemunculan kata dalam dokumen (*term frequency*).

## Vektor

Vektor adalah kajian aljabar yang biasanya dapat digunakan untuk memecahkan permasalahan fisika seperti gerak, gaya, dan lainnya. Tetapi, vektor yang digunakan pada *information retrieval* ini adalah sebagai kajian aljabar. Secara geometri vektor adalah ruas garis berarah. Vektor memiliki ukuran panjang dan juga arah.

## Cosine similarity

*Cosine similarity* adalah ukuran kemiripan antara dua buah vektor bukan nol dari ruang hasilkali dalam. Nilainya didefinisikan sebagai kosinus sudut antara kedua vektor tersebut. Berikut adalah rumus untuk menemukan *cosine similarity* dari dua buah vektor.

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

## Bab 3

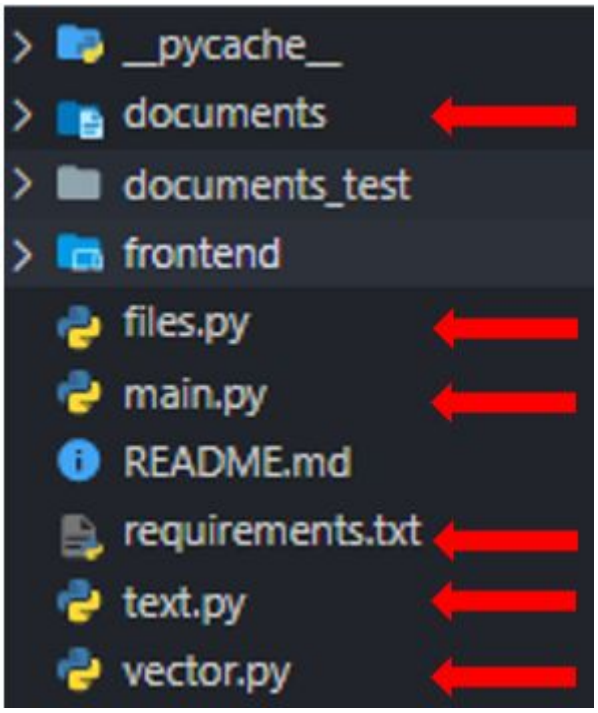
Pembuatan *search engine* berbasis web ini dilakukan dengan membaginya menjadi dua bagian, *frontend* dan *backend*. *Frontend* bertugas untuk menampilkan data dengan tampilan yang menarik dan ramah bagi *user*, tidak ada algoritma atau proses perhitungan sama sekali di *frontend*. Secara garis besar *backend* bertugas untuk melakukan segala macam perhitungan dan juga me-*manage* file atau documents yang digunakan sebagai salah satu input pada salah satu proses perhitungan dalam *search engine*, baik yang dimasukkan secara manual ke dalam *directory backend* ataupun yang diinput oleh pengguna dari frontend.

Secara garis besar cara kerja dari integrasi antara *frontend* dan *backend* adalah sebagai berikut. *Frontend* akan menampilkan tampilan kepada pengguna di web, serta menyediakan kolom input bagi pengguna untuk memasukkan file ataupun *search query*. Ketika seorang pengguna meng-*input search query* dan meng-*click search*, maka *frontend* akan mengirimkan *HTTP request* ke *backend,* untuk kasus ini, *method* yang digunakan adalah *post method*, lebih detailnya akan dijelaskan pada bagian selanjutnya. Saat *backend* menerima *HTTP request* dari *frontend*, maka *backend* akan mulai melakukan perhitungan atau algoritma untuk *request* tersebut. Hasil dari perhitungan ataupun kalkulasi dikirim kembali sebagai *response* ke *frontend* dalam format *JSON*. Setelah mendapatkan *response* dari *backend*, maka *frontend* akan menggunakan data itu baik untuk *logic* lainnya ataupun ditampilkan ke pengguna.

### 3.1 Backend

*Framework* yang digunakkan untuk mengimplementasikan *backend* adalah Flask. Flask adalah suatu kerangka kerja web ringan dan mikro yang berbasis bahasa Python. Pertimbangan kelompok kami memilih Flask adalah karena kami berencana untuk melakukan perhitungan dan kalkulasi yang rumit dengan menggunakan Python, karena bahasanya yang mudah digunakkan dan banyaknya fungsi ataupun utilitas bawaan yang dapat memudahkan perhitungan, sehingga untuk *simplicity* kelompok kami setuju untuk melakukan perhitungan yang rumit tersebut dan *backend services* pada tempat yang sama yaitu di *backend*.

### 3.1.1 Struktur folder dan file




- Folder documents : tempat menyimpan dokumen baik yang dimasukkan secara manual ataupun yang berasal dari input user dari *frontend*.
- files.py : segala *logic* yang berhubungan dengan *files*, mulai dari mengambil nama - nama files yang ada, mengubah suatu *file txt* menjadi *string*, dan *Class Document*, yang digunakkan sebagai representasi dari sebuah dokumen.
- main.py : *script* utama yang akan dijalankan untuk *backend*, terdapat konfigurasi dari Flask, implementasi *api*, serta fungsional untuk menjalankan *backend server*.
- text.py : segala *logic* yang berhubungan dengan *text* atau *string*, yang merupakan inti dari perhitungan dan kalkulasi.
- vector.py : segala *logic* yang berhubungan dengan perhitungan dan kalkulasi vektor, merupakan inti dari perhitungan dan kalkulasi, terdapat juga metode pencarian similaritas dengan *cosine similarity* yang diimplementasikan menggunakan vektor.

### 3.1.2 vector.py

Pada implementasinya untuk perhitungan dan kalkulasi vektor dengan menggunakan bahasa Python, diperlukan untuk meng-*import* modul atau *library math*. Pada kasus ini kegunaannya adalah untuk melakukan operasi akar pangkat dua. Vektor diimplementasikan dengan menggunakan struktur data *array*.

**dot(v1, v2)**

```
def dot(v1, v2):
    n = len(v1)
    result = 0
    for i in range(n):
        result += (v1[i] * v2[i])

    return result
```

Ini adalah fungsi yang digunakkan untuk menghitung *dot product* dari dua buah vektor. Parameter dari fungsi ini adalah v1 dan v2 dengan tipe data *array*. Perhitungannya cukup sederhana, yaitu dengan menjumlahkan setiap elemen pada v1, dan v2 yang indeksnya sama. Nilai yang di-*return* bertipe *integer*.

**norm(v)**

```
def norm(v):
    result = 0
    for el in v:
        result += el**2

    return math.sqrt(result)
```

Ini adalah fungsi yang digunakkan untuk menghitung norma atau magnitudo dari sebuah vektor. Parameter dari fungsi ini adalah v yang bertipe data *array*.

Perhitungannya adalah hasil akar pangkat dua dari semua kuadrat dari elemen v. Nilai yang di-*return* bertipe *float.*

**sim(v1, v2)**

```
def sim(v1, v2):
    n1 = norm(v1)
    n2 = norm(v2)

    return (dot(v1, v2)/(n1*n2))
```

Ini adalah fungsi yang digunakkan untuk menghitung similaritas dengan metode *cosine similarity*. Fungsi ini menerima dua parameter, v1 dan v2, dengan tipe data *array*. Cara menghitungnya cukup sederhana, yaitu *dot product* dari v1 dan v2 dibagi dengan norma v1 dikali norma v2. Nilai yang di-*return* bertipe *float*.

### 3.1.3 text.py

Pada bagian ini adalah segala fungsi dan *logic* yang berhubungan dengan manipulasi *string* atau *text*. Untuk implementasinya diperlukan untuk *import library* Sastrawi. Sastrawi digunakkan untuk melakukan penghapusan *stop words* dan melakukan proses *stemming* (mengubah suatu kata ke kata dasarnya) untuk bahasa Indonesia.

**clean_text(text)**

```
def clean_text(text):

    # removing punctuation
    for c in string.punctuation:
        text = text.replace(c, "")

    # removing excessive whitespace
    text = " ".join(text.split())
```

```
    # text to array of word
    words = text.split()

    # removing stopwords
    words = [word for word in words if word not in stopwords]

    # stemming word in query
    words = [stemmer.stem(word) for word in words]

    return words
```

Fungsi ini digunakkan untuk melakukan pembersihan terhadap suatu teks sehingga layak untuk digunakkan dalam perhitungan. Parameter dari fungsi ini adalah *text* yang bertipe *string*. Terdapat beberapa tahap yang dilakukan untuk membersihkan teks. Pertama kita akan menghilangkan segala macam tanda baca yang terdapat di teks. Proses selanjutnya adalah menghilangkan *whitespace* yang berlebihan. Lalu, mengubah teks ke dalam tipe data *array of string* yang dimana setiap elemennya adalah kata pada teks. Kemudian, kita harus menghilangkan *stopwords* dari kumpulan kata tersebut. Proses terakhirnya adalah melakukan *stemming* pada setiap kata yang ada pada *array* tersebut. Nilai yang di-*return* adalah *array of string* yang elemennya merupakan kata kata dari teks yang sudah dibersihkan.

**get_first_sentence(text)**

```
def get_first_sentence(text):
    firstsentence = ""
    i = 0
    while(text[i] not in [".", "!", "?"]):
        firstsentence += text[i]
        i += 1
    firstsentence += text[i]

    return firstsentence
```

Fungsi ini digunakkan untuk mendapatkan kalimat pertama dari sebuah teks. Parameter dari fungsi ini adalah *text* yang bertipe *string*. Algoritmanya adalah dengan melakukan

*while loop* sampai kita menemukan tanda baca tertentu, dimana di setiap pengulangannya akan meng-*concat* karakter yang dibaca ke variable *firstsentence*. Nilai yang di-*return* adalah *string* yang merupakan kalimat pertama dari sebuah teks.

**term_freq(docs)**

```
def term_freq(docs):
    result = [{} for doc in docs]
    for i in range(len(result)):
        words = clean_text(docs[i])
        for word in words:
            for j in range(len(result)):
                if j == i:
                    if word not in result[j]:
                        result[j][word] = 1
                    else:
                        result[j][word] += 1
                else:
                    if word not in result[j]:
                        result[j][word] = 0

    return result
```

Fungsi ini digunakkan untuk membuat ‘tabel’ dan menyimpan nilai frekuensi dari kata - kata yang ada pada seluruh dokumen di *documents* / *docs*. Parameter dari fungsi ini adalah *docs*, yaitu *array of string*. Secara garis besar, cara kerja dari algoritma ini adalah kita mengiterasi untuk setiap kata untuk setiap dokumen, untuk setiap kata kita mengecek apakah term tersebut sudah ada pada ‘tabel’, jika kita sedang berada pada suatu dokumen pada proses iterasi, maka kita akan *increment* nilai dari baris *term* jika sudah ada barisnya, jika belum ada maka akan diinisiasi dengan nilai sama dengan 1, jika diluar dokumen maka, hanya akan menginisiasi baris baru dengan nilai sama dengan 0 jika belum ada baris dengan *term* tersebut. Nilai yang di-*return* adalah *array of dictionary* yang dimana setiap *dictionary* tersebut adalah frekuensi dari *term*.

**dict_to_vector(dic)**

```
def dict_to_vector(dic):
    vector = [dic[k] for k in dic.keys()]
    return vector
```

Fungsi ini digunakkan untuk mengubah sebua *dictionary* menjadi vektor yang bertipe *array of integer*. Parameter dari fungsi ini adalah *dic* dengan tipe *dictionary*. Nilai yang di-*return* adalah bertipe *array of integer*.

### 3.1.4 files.py

Disini terdapat segala *logic* yang berhubungan dengan *files* juga konfigurasi untuk *folder* yang digunakkan untuk menyimpan dokumen. Untuk implementasinya diperlukan *import os*, yang digunakkan untuk mengambil lokasi dari file ini, menglist *file* pada suatu directory, ataupun menggabungkan *path*.

**file_to_string(file_path)**

```
def file_to_string(file_path):
    data = ""
    exact_path = os.path.relpath(file_path, CURRENT_PATH)
    with open(exact_path, "r", encoding="utf8") as file:
        data = file.read().replace("\n", " ")

    return data
```

Fungsi ini digunakan untuk mengubah suatu file menjadi *string*. Parameter dari fungsi ini adalah *file_path* dengan tipe data *string*. Implementasinya cukup sederhana, kita akan menghubungkan *file_path* dengan *path* sekarang dari file ini. Nilai yang di-*return* adalah bertipe *string*.

**get_filenames()**

```
def get_filenames():
    return os.listdir(DOCUMENT_DIRECTORY)
```

Fungsi ini mengembalikan semua file yang ada pada *folder* penyimpanan dokumen.

**class Document**

```
class Document:
    def __init__(self, filename):
        self.filename = filename
        self.title = (filename.split("."))[0]
        self.content = file_to_string(
            f".\\{DOCUMENT_DIRECTORY}\\{self.filename}")
        self.term_freq = {}
        self.similarity = 0
        self.length = len(clean_text(self.content))
        self.first_sentence = get_first_sentence(self.content)
```

*Class* ini digunakan untuk mendefinisikan suatu objek bertipe *Document*. *Class* ini memiliki beberapa atribut sebagai berikut.

- filename : nama dari *file*.
- title : judul dari *file* yang merupakan nama dari *file* tersebut itu sendiri tetapi tanpa *extension*.
- content : isi dari *file* yang didapati dengan menggunakan fungsi *file_to_string* yang sudah didefinisikan sebelumnya di files.py.
- term_freq : *dictionary* dimana *key*-nya merupakan kata atau *term* dan *value*-nya adalah frekuensi kemunculan.
- similarity : tingkat kesamaan terhadap *search query,* nantinya akan dihitung dengan menggunakan metode *cosine similarity*.
- length : panjang dari *content*, yaitu jumlah kata pada *file*.
- first_sentence : kalimat pertama dari *file.*

### 3.1.5 main.py

Ini adalah script utama yang dijalankan pada *server backend*, terdapat konfigurasi dari Flask, implementasi *api*, serta fungsional untuk menjalankan *backend server*. Untuk implementasinya ada beberapa *library* yang harus di *import*, diantaranya adalah *json, os, flask,* dan modul modul utilitas yang sudah didefinisikan di atas seperti *vector.py, text.py,* dan *files.py*. Modul *json* digunakkan karena yang akan di return sebagai *response* dari *api* yang ada adalah dalam format *json*.

**Konfigurasi**

```
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = DOCUMENT_DIRECTORY
CORS(app)
```

Konfigurasi yang dilakukan disini adalah menginisiasi app dengan flask, mengatur tempat kita mengupload dokumen, serta menggunakan CORS pada aplikasi flask ini.

**/search**

```
@app.route("/search", methods=["POST"])
def search_documents():

    query = request.get_json()["query"]
    print(query)

    res = {
        "docs": [Document(filename).__dict__ for filename in get_filenames()],
        "query": {
            "input": query,
            "term_freq": []
        }
    }

    freq = term_freq([doc["content"] for doc in res["docs"]] + [query])

    for i in range(len(freq)):
        if i == len(freq) - 1:
            res["query"]["term_freq"] = freq[i]
        else:
            res["docs"][i]["term_freq"] = freq[i]

    for doc in res["docs"]:
        doc["similarity"] = sim(dict_to_vector(
            res["query"]["term_freq"]), dict_to_vector(doc["term_freq"]))

    res["docs"] = sorted(
        res["docs"], key=lambda k: k["similarity"], reverse=True)

    return jsonify(res)
```

Ini adalah fungsi yang menangani *api* dengan *route "/*search*"*. Untuk *api* ini, digunakan *POST method*, jadi terdapat data yang ada pada *request body*. *Request body* memiliki format *json*, dimana terdapat suatu *key* "query" yang berisi *search query* yang dimasukkan pengguna. Kita pertama - tama mengambil *query* yang dimasukkan pengguna. Lalu menginisiasikan bentuk dari *response* yang akan kita kembalikan seperti pada gambar di atas. Lalu kita menghitung frekuensi dari setiap kata pada setiap dokumen dan *query* pengguna. Kemudian kita isi atribut "term_freq" dari setiap dokumen yang ada pada *array of Document* seperti diatas. Kemudian kita akan menghitung nilai similaritas dari setiap dokumen. Terakhir kita akan mengurutkan *array of Document* tersebut dengan menurun.

**/doc/<string:filename>**

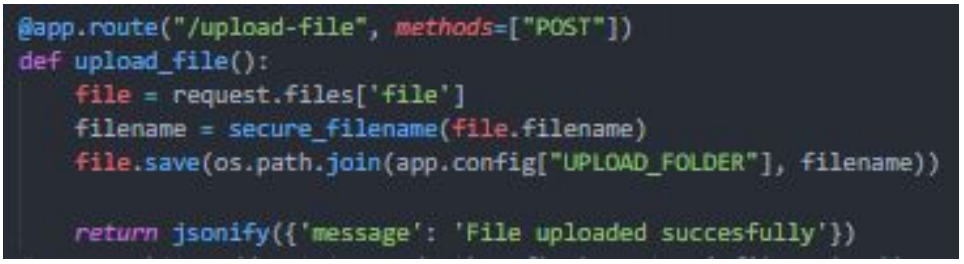
```
@app.route("/doc/<string:filename>", methods=["GET"])
def get_document(filename):

    doc = Document(filename)

    return jsonify(
        {"doc":
            {
                "title": doc.title,
                "content": doc.content
            }
        })
```

Ini adalah fungsi yang digunakan untuk menangani *api* dengan *route* seperti pada subjudul diatas. *Route* dari *api* ini memiliki parameter *filename* yang bertipe *string*. *Api* ini menggunakan *GET method*, digunakan untuk mengambil judul dan isi dokumen dari *filename* pada parameter.

**/upload-file**



```python
@app.route("/upload-file", methods=["POST"])
def upload_file():
    file = request.files['file']
    filename = secure_filename(file.filename)
    file.save(os.path.join(app.config["UPLOAD_FOLDER"], filename))

    return jsonify({'message': 'File uploaded succesfully'})
```

Ini adalah fungsi yang digunakan untuk menangani *api* dengan *route* "/upload-file". *Api* ini menggunakan *POST method* yang dimana pada *request body* yang memiliki format *json* terdapat *key* "file" yang *value*-nya adalah input *file* dari pengguna. Disini implementasinya mudah, yaitu dengan meng-*save file* dengan metode save yang dimiliki objek bertipe *file*, pada lokasi atau *path* berdasarkan konfigurasi diatas. *Api* ini akan mengembalikkan pesan berhasil.

### 3.2 Frontend

Implementasi dari *frontend* adalah menggunakan *framework* React. Satu hal penting yang perlu diperhatikan adalah penggunaan *axios* untuk melakukan *api call* ke *backend* untuk mendapatkan hasil dari *search query*, mengambil isi dari suatu dokumen, dan *file upload*.

## Bab 4
## Eksperimen

Pada bab ini, hasil eksekusi program terhadap beberapa contoh kasus akan dijelaskan. Contoh kasus akan meliputi search query yang ditentukan oleh tim penulis. Mengingat ada 4 topik utama dari dokumen yang sudah tersedia yaitu paleontologi, *capital punishment*, *video games*, dan *student exchange*, query yang akan diuji disesuaikan dengan keempat topik tersebut. Berikut penjelasan spesifik mengenai setiap query.

### 4.1 Search Query 1: Fosil

*Search query* yang akan digunakan pertama adalah dengan memasukan kata kunci 'fosil' ke *search engine*. Hal ini dilakukan untuk menguji kemampuan *search engine* untuk mencari suatu kata kunci. Hasil yang diharapkan adalah *search engine* menampilkan dokumen-dokumen paleontologi di urutan-urutan awal. Berikut hasil eksperimen ini.

Gambar 4.1. Hasil *Search Query* ‘fosil’

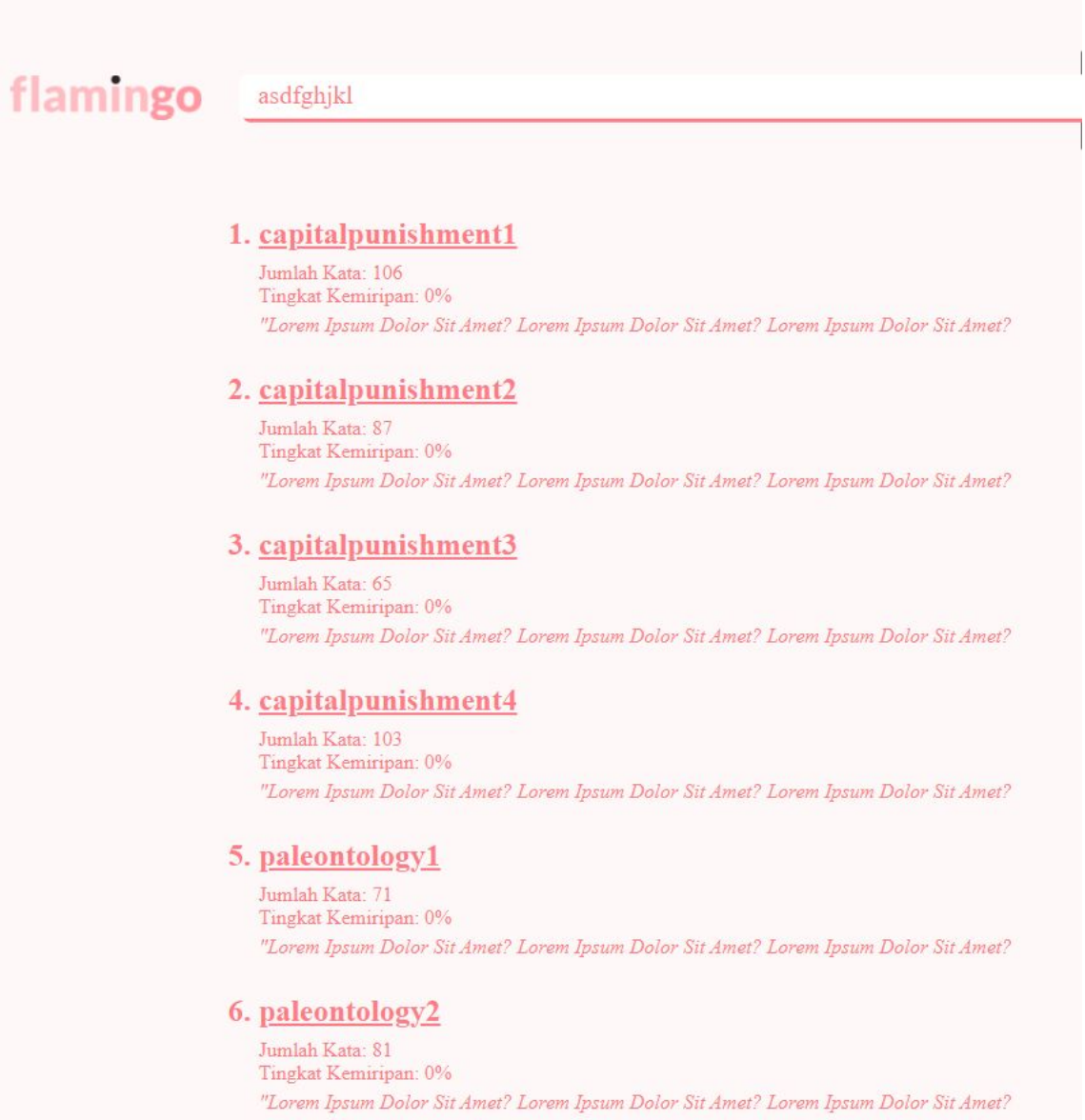
Sesuai yang dengan yang diharapkan, empat urutan pertama dari hasil *search* yang dikeluarkan *search engine* adalah dokumen mengenai paleontologi. Hal ini berarti bahwa algoritma kemiripan kata bekerja dengan sesuai. Seperti yang dapat dilihat juga, program mengurutkan hasil *search* berdasarkan tingkat kemiripan. Kemiripan juga dapat dilihat dari jumlah kata yang sesuai dengan *search query*. Berikut hasilnya.


Gambar 4.2. *Search Term* Fosil

Dari kiri ke kanan, dapat dilihat term ‘fosil’. Kolom pertama adalah jumlah kata ‘fosil’ pada *search query* yaitu 1. Kolom berikutnya adalah jumlah kata ‘fosil’ pada dokumen di urutan 1,2,3..., dst. Seperti yang dapat dilihat, dokumen dengan jumlah kata paling banyak yang paling serupa dengan kata pada *search query* (yang dalam kasus ini adalah ‘fosil’) diletakkan pada urutan-urutan pertama. Dari hasil query pertama ini, didapatkan bahwa *search engine* berjalan sesuai dengan algoritma kemiripan (*similarity*) yang diterapkan untuk pencarian informasi.

**4.2 Search Query: asdfghjkl**

*Search query* yang digunakan selanjutnya adalah kumpulan huruf yang tidak bermakna yaitu ‘asdfghjkl’. Hal ini dilakukan untuk menguji kemampuan *search engine* saat menerima input berupa suatu karakter yang pasti tidak memiliki kemiripan pada seluruh dokumen yang tersedia. Hasil yang diharapkan dari *search engine* adalah tidak ada dokumen yang memiliki kemiripan dengan *search query* ini. Berikut hasil dari *search engine*.




Gambar 4.3. Hasil *Search Query* ‘asdfghjkl’

Seperti yang dapat dilihat dari hasil *search query* di atas, dapat terlihat bahwa semua dokumen memberikan kemiripan sebesar 0% terhadap *search query*. Walau begitu, semua dokumen tetap ditampilkan oleh *search engine*. Perlu dicatat bahwa dokumen diurutkan berdasar alfabet huruf pertama dari judul dokumen. Hal ini dapat disebabkan oleh pengaturan pada folder penyimpan dokumen. Dalam folder tersebut memang setiap dokumen diatur secara alfabetik, dan hasil bacaan dokumen tersebut

ditampilkan oleh *search engine*. Hal ini juga didukung oleh tabel *search term* yang menghasilkan 0 untuk semua list term yang ada dalam dokumen, hal ini berarti tidak satupun *term* sesuai dengan *search query*. Berikut potongan dari list *search term* yang menghasilkan 0 untuk semua *term*.

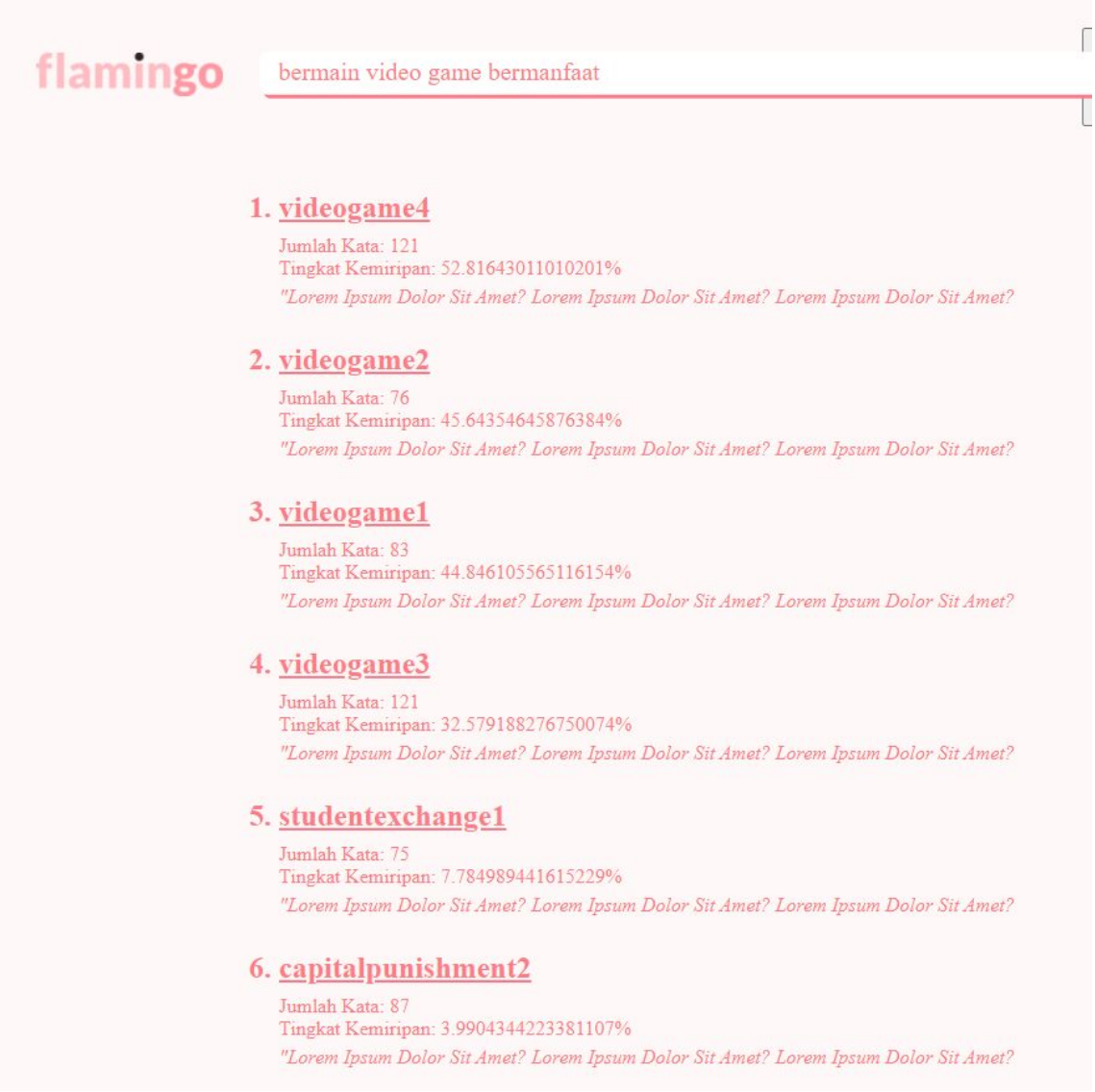
| asdfghjkl | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Gambar 4.4 *Tabel Search Term* untuk ‘asdfghjkl’

Seperti yang dapat dilihat, *search term* hanya bernilai 1 di kolom pertama yang adalah kolom *search query*. Di kolom lainnya, semua menghasilkan angka 0 karena tidak ada satupun yang sesuai dengan ‘asdfghjkl’. Dalam kasus ini, walau *search engine* tidak menghasilkan suatu pesan seperti “*search* tidak ditemukan”, seperti Google dan semacamnya, namun search engine tetap berjalan dengan benar karena menampilkan similaritas 0 di semua dokumen. Untuk kedepannya, hal ini dapat dikembangkan lebih lanjut.

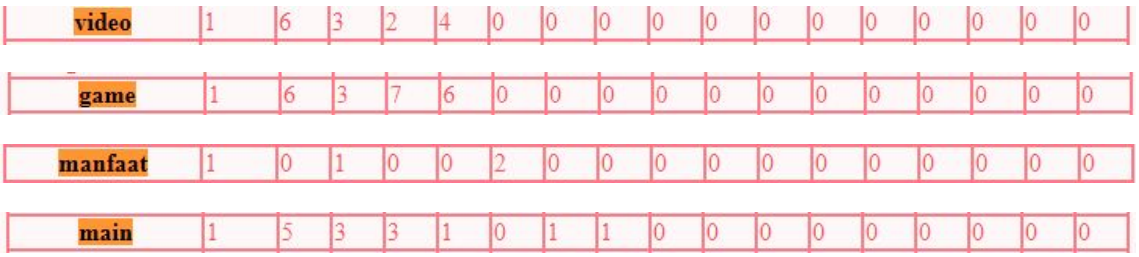
**4.3 Search Query: bermain video game bermanfaat**

*Search query* yang digunakan selanjutnya adalah kalimat ‘bermain video game bermanfaat’. Input ini digunakan untuk menguji kemampuan *search engine* dalam menerima input berupa kalimat. Hasil yang diharapkan dari *search engine* adalah artikel mengenai video game yang sesuai dengan *search query*. Berikut adalah hasil dari *search engine*.




Gambar 4.5 Hasil *Search Query* ‘bermain video game bermanfaat’

Berdasar hasil dari *search engine*, dapat terlihat bahwa artikel dengan tema utama video game terdapat pada 4 urutan pertama dari hasil *search query*. Selain itu, perlu dicatat bahwa artikel videogame4 terletak pada urutan pertama dengan kemiripan 52.8% yang adalah kemiripan paling tinggi secara keseluruhan. Hal ini disebabkan karena artikel tersebut mengandung term paling banyak dari *search query* yaitu ‘bermain video game bermanfaat’. Daftar *term* untuk dokumen ini juga mendukung hal tersebut seperti ditunjukan sebagai berikut.




Gambar 4.6. Daftar *Term*

Seperti yang dapat dilihat diatas, kolom kedua adalah untuk dokumen urutan pertama yaitu videogame4. Dapat dilihat jumlah kata paling banyak yang sesuai dengan search term terdapat di dokumen tersebut. Hal yang menarik yang dapat dilihat dari daftar *term* tersebut adalah untuk kata manfaat dan main dapat terlihat juga di dokumen ke-5 dan ke-6. Hal ini dapat terjadi karena dokumen ke-5 dan ke-6, yang dalam kasus ini adalah studentexchange1 dan capitalpunishment2 mengandung kata tersebut dalam teks dokumen. Naamun, karena sedikitnya kemiripan bila dibandingkan dengan term lain, maka kedua dokumen ini diletakkan di urutan dibawah dokumen mengenai video game. Walau begitu, perlu dicatat bahwa dokumen juga memiliki kemiripan yang disebabkan adanya kata manfaat dan main tadi.

**Bab 5**

**Kesimpulan, Saran dan Refleksi**

## 5.1 Kesimpulan

Tim penulis dapat menyimpulkan bahwa, setelah mengerjakan tugas besar II mata kuliah IF2123 Aljabar Linier dan Geometri, pemrograman web merupakan cabang ilmu yang sangat menarik dan berupa multi-faset. Terdapat berbagai cara untuk mencapai satu tujuan solusi, sehingga pengerjaan tugas besar ini selalu diselingi dengan perasaan dan pengalaman positif. Baik dari segi *backend* maupun *frontend*, proses realisasi yang dimulai dari eksplorasi tools yang tersedia, belajar tentang penggunaan *Flask* hingga implementasi *frontend* dari mockup prototipe antarmuka-pengguna, sangatlah berkesan dan bisa dikatakan membangkitkan jiwa petualang teknologi di dalam diri tim penulis. Adapula pokok bahasan dari tugas besar ini yakni aplikasi *dot product* atau perkalian titik pada sistem temu-balik informasi sangatlah menarik, dan setelah riset lebih lagi, memberi kami gambaran terhadap teknologi dan metode yang digunakan dalam aplikasi-aplikasi yang digunakan tiap hari-nya seperti *Google, Youtube,* dan *Duckduckgo.*

Kami juga berhasil memelajari dan menyimpulkan beberapa pelajaran-pelajaran berikut yang kami dapatkan selama pengerjaan tugas besar ini, sebagai berikut:

1. Dalam pengerjaan tugas besar, maupun tugas lainnya dalam organisasi atau pekerjaan nantinya, dibutuhkan komunikasi yang baik dan lancar sehingga pengerjaan proyek/tugas bisa berjalan tanpa hambatan dan tanpa konflik dari sisi manapun, serta akan menghasilkan kolaborasi yang maksimal diantara semua kontributor proyek/tugas.
2. Sangat dibutuhkannya jiwa eksplorasi serta keinginan untuk memelajari hal-hal baru dalam proyek-proyek serta pekerjaan yang akan kita ambil di masa depan, sehingga harus dikembangkan keinginan tersebut sejak dini, dan gagas ini pula yang sangat didorong oleh kurikulum pada program studi Teknik Informatika di Institut Teknologi Bandung.
3. Kurva pembelajaran pemrograman web pada awal terlihat sangatlah landai dikarenakan betapa mudahnya kita dapat sampai ke titik dimana kita bisa mulai

merealisasikan situs web yang ada di dalam benak. Namun bidang pemrograman web ini membutuhkan banyak latihan untuk mengakrabkan diri dengan banyak *framework-framework* yang tersedia, serta cara untuk mengembangkan situs-situs dengan praktik terbaik.

### 5.2 Saran

Tim penulis ingin menyampaikan beberapa saran untuk tim pengampu serta tim asisten mata kuliah IF2123 Aljabar Linier dan Geometri yakni diadakannya sesi asistensi-singkat, dengan pertimbangan tugas besar mata kuliah ini tidak se-*intens* dan durasinya hanya kira-kira setengah durasi tugas besar lain, walau kelompok tugas besar dapat kapanpun mengajukan pertanyaan pada asisten, ada baiknya apabila sebuah sesi asistensi-singkat yang formal bisa diadakan demi kelancaran pengerjaan dan menambahkan motivasi kepada kelompok-kelompok yang sedang mengerjakan tugas besar.

Menurut kami, tugas besar II mata kuliah IF2123 Aljabar Linier dan Geometri ini sudah berjalan dengan sangat baik dan efektif, baik dalam segi pelaksanaan, pertimbangan beban kerja terhadap tugas mata kuliah lain, juga dalam segi ilmu yang dipelajari. Sehingga dengan itu penulis juga ingin menutup pernyataan saran ini dengan apresiasi sebesar-besarnya terhadap tim pengampu dan asisten mata kuliah IF2123 ini.

### 5.3 Refleksi

Setelah pengerjaan tugas besar ini, para penulis menyadari pentingnya peran tugas besar, terutama pada program studi kami, yang sisi tekniknya sangat erat dengan praktik dan pengaplikasian secara langsung. Tak hanya berguna untuk mengembangkan kemampuan serta ilmu yang terlibat dalam tugas besar itu sendiri, namun penerapan sistem pengerjaan menggunakan *version control* melalui git dan github akan membantu nantinya dalam pengembangan portofolio kami untuk masa depan saat sedang meniti karir. Tugas besar ini juga merupakan media yang seolah-olah ‘memaksa’ kami untuk mengembangkan kemampuan interpersonal yakni kemampuan berkomunikasi dengan efektif yang nantinya akan sangat dibutuhkan baik sebagai *engineer* yang bekerja dalam

tim dalam lingkungan korporat, maupun seorang *entrepreneur* yang mengembangkan usahanya sendiri dengan sekutu ataupun karyawannya.