

## Laporan Tugas Besar 2

### IF2123 Aljabar Linier dan Geometri

EIGENPUSTAKA: SISTEM PERPUSTAKAAN DIGITAL DENGAN REKOMENDASI SEMANTIK DAN PENCARIAN VISUAL BERBASIS SVD

Semester I Tahun 2025/2026



Disusun oleh:

13524122 Nathaniel Christian

13524139 Azri Arzaq Pohan

13524142 Rasyad Satyatma

LABORATORIUM ILMU DAN REKAYASA KOMPUTASI  
PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG

## Daftar Isi

<b>1 Pendahuluan</b>	<b>2</b>
<b>2 Dasar Teori</b>	<b>3</b>
2.1 Sistem Temu Balik Gambar berbasis PCA . . . . .	3
2.1.1 Persiapan Data Gambar . . . . .	3
2.1.2 Dekomposisi SVD dan Eigencovers . . . . .	3
2.1.3 Proyeksi ke Ruang Fitur . . . . .	3
2.1.4 Pencarian Gambar . . . . .	3
2.2 Sistem Rekomendasi dan Temu Balik Teks berbasis Analisis Semantik Laten (LSA) . . . . .	4
2.2.1 Persiapan Data Dokumen . . . . .	4
2.2.2 Matriks Term–Document . . . . .	4
2.2.3 Pembobotan TF–IDF . . . . .	5
2.2.4 Reduksi Dimensi dengan SVD . . . . .	5
2.2.5 Representasi Dokumen dalam Ruang Semantik Laten . . . . .	5
2.2.6 Metode Similaritas . . . . .	6
2.3 Teori Singkat dan Jawaban . . . . .	6
2.3.1 Eigenvalue dan Eigenvector . . . . .	6
2.3.2 Singular Value Decomposition . . . . .	6
2.3.3 Jawaban . . . . .	7
<b>3 Desain dan Implementasi</b>	<b>8</b>
3.1 Arsitektur Aplikasi ( <i>Root</i> ) . . . . .	8
3.2 Arsitektur Aplikasi ( <i>Front End</i> ) . . . . .	8
3.3 Arsitektur Aplikasi ( <i>Back End</i> ) . . . . .	10
3.3.1 Kelas-Kelas Penting . . . . .	10
3.4 Spesifikasi Bonus . . . . .	11
3.4.1 Kontainerisasi dengan Docker dan Deployment . . . . .	11
3.4.2 Beautiful UI/UX . . . . .	12
3.4.3 Search by Document . . . . .	12
<b>4 Eksperimen</b>	<b>14</b>
4.1 Eksperimen pada Laman Utama . . . . .	14
4.2 Eksperimen pada Laman Detail Buku . . . . .	14
4.3 Eksperimen Pencarian Menggunakan Judul . . . . .	15
4.4 Eksperimen Pencarian Menggunakan Upload Image (dari asisten) . . . . .	16
4.5 Eksperimen Pencarian Menggunakan Upload Image . . . . .	18
4.6 Eksperimen Rekomendasi . . . . .	19
4.7 Eksperimen Pencarian Menggunakan Upload Document (bonus) . . . . .	20
<b>5 Penutup</b>	<b>22</b>
5.1 Kesimpulan . . . . .	22
5.2 Saran . . . . .	22
5.3 Refleksi . . . . .	22
<b>6 Lampiran</b>	<b>24</b>

## 1 Pendahuluan

Di Desa Konohagakure, hiduplah dua orang mahasiswa Teknik Informatika bernama Ayu dan Asep Kasep. Akhir-akhir ini, Ayu merasa kewalahan memahami materi dekomposisi matriks yang diajarkan dalam mata kuliah IF2123 Aljabar Linier dan Geometri karena ketimbang memperhatikan penjelasan dosen di depan, ia malah asyik doomscrolling di media sosial. Supaya bisa bertanggung jawab kepada kedua orang tuanya yang telah membayai kuliahnya, Ayu pun bertekad untuk memperbaiki situasi dengan belajar ke perpustakaan. Sesampainya di sana, ia takjub akan banyaknya koleksi buku yang tersedia. Sejak saat itu, Ayu pun gemar mengunjungi perpustakaan untuk mengeksplorasi berbagai macam buku yang menarik minatnya.

Di sisi lain, Asep adalah seorang mahasiswa yang bekerja paruh waktu sebagai pustakawan di perpustakaan kampus. Sejak Ayu mulai rajin berkunjung ke perpustakaan, Asep sering membantunya mencari buku-buku yang diinginkannya. Awalnya, Ayu mengira Asep adalah seorang performative male yang hanya ingin pamer di hadapannya. Apalagi, Asep sering sekali minum matcha. Namun, seiring berjalaninya waktu, Ayu menyadari bahwa Asep adalah sosok yang cerdas dan tulus dalam membantu sesama. Asep juga sadar bahwa Ayu adalah pribadi yang tekun dan pantang menyerah. Akhirnya, mereka pun jatuh cinta.



Gambar 1: Anggaplah ini Ayu dan Asep.

Sumber: [harrypotter.com – The Most Memorable Moments in the Hogwarts Library](https://harrypotter.com/the-most-memorable-moments-in-the-hogwarts-library)

Asep menyadari bahwa Ayu sering kali kesulitan menemukan buku yang diinginkannya atau melanjutkan bacaan dari buku yang pernah dipinjamnya sebelumnya. Hal ini dikarenakan sistem pencarian buku di perpustakaan yang masih konvensional dan kurang efisien. Melihat hal tersebut, Asep terinspirasi untuk mengembangkan sebuah sistem temu balik informasi berbasis singular value decomposition (SVD) dan vektor eigen yang dapat membantu Ayu dan pengunjung perpustakaan lainnya dalam menemukan buku dengan lebih mudah dan cepat.

Untuk membuktikan bahwa mahasiswa Teknik Informatika memanglah solid, mari kita bantu Asep dalam merancang dan mengimplementasikan sistem temu balik informasi tersebut dengan memanfaatkan konsep-konsep aljabar linier yang telah kita pelajari di mata kuliah IF2123 Aljabar Linier dan Geometri.

## 2 Dasar Teori

### 2.1 Sistem Temu Balik Gambar berbasis PCA

#### 2.1.1 Persiapan Data Gambar

Sebelum metode PCA dapat diterapkan pada kumpulan gambar, seluruh citra perlu dipersiapkan agar berada dalam bentuk yang konsisten dan dapat dianalisis secara matematis.

Gambar berwarna diubah menjadi citra *grayscale* menggunakan kombinasi linear kanal RGB dengan bobot luminansi standar:

$$I(x, y) = 0.2126 R(x, y) + 0.7152 G(x, y) + 0.0722 B(x, y).$$

Citra *grayscale* kemudian dinormalisasi sehingga intensitas berada pada rentang  $[0, 1]$ . Seluruh gambar memiliki resolusi tetap  $h \times w$  pixel, sehingga setiap gambar dapat direpresentasikan sebagai vektor berdimensi  $d = h \cdot w$ .

Jika terdapat  $n$  gambar, maka seluruh citra dihimpun dalam matriks:

$$X = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{bmatrix} \in R^{n \times d}.$$

Dihitung *mean image*:

$$\mu = \frac{1}{n} \sum_{k=1}^n x_k,$$

dan setiap vektor citra dipusatkan:  $\tilde{x}_k = x_k - \mu$ .

#### 2.1.2 Dekomposisi SVD dan Eigencovers

SVD diterapkan pada matriks  $\tilde{X}^T \in R^{d \times n}$ :

$$\tilde{X}^T = U \Sigma V^T,$$

di mana kolom-kolom  $U \in R^{d \times n}$  merupakan *eigencovers*, yaitu basis ortonormal yang merepresentasikan variasi visual utama pada kumpulan gambar.

#### 2.1.3 Proyeksi ke Ruang Fitur

Setiap gambar diproyeksikan ke ruang *eigencovers*:

$$c_k = U^T \tilde{x}_k.$$

Koefisien kemudian dinormalisasi untuk pencarian berbasis *cosine similarity*:

$$\hat{c}_k = \frac{c_k}{\|c_k\|}.$$

#### 2.1.4 Pencarian Gambar

Untuk gambar query  $q$ , dihitung vektor terpusat  $\tilde{q} = q - \mu$  dan koefisien proyeksinya:

$$\hat{c}_q = \frac{U^T \tilde{q}}{\|U^T \tilde{q}\|}.$$

Kemiripan dengan setiap gambar dalam database dihitung menggunakan *cosine similarity*:

$$\text{sim}(q, x_k) = \hat{c}_q \cdot \hat{c}_k.$$

## 2.2 Sistem Rekomendasi dan Temu Balik Teks berbasis Analisis Semantik Laten (LSA)

Latent Semantic Analysis (LSA) adalah metode untuk menemukan hubungan semantik tersembunyi antara kata dan dokumen dalam sebuah korpus atau kumpulan teks. LSA bekerja dengan membangun matriks term-document dari seluruh dokumen, kemudian melakukan reduksi dimensi menggunakan Singular Value Decomposition (SVD) untuk menemukan *topic space* (semacam pola tersembunyi) yang tidak terlihat melalui frekuensi kata mentah.

### 2.2.1 Persiapan Data Dokumen

Sebelum LSA diterapkan, seluruh dokumen harus melalui preprocessing agar representasinya lebih bersih dan konsisten. Langkah-langkahnya adalah:

1. Normalisasi: pembersihan karakter non-alphabet, huruf kecil, dan split kalimat
2. Tokenisasi menggunakan library NLTK
3. Pembersihan stopwords
4. Stemming menggunakan Porter Stemmer dari library NLTK

Hasil akhir tahap ini adalah list token dari masing-masing dokumen.

Kode program dapat ditemukan di `utils.py`, dan di `lsa.py`, khususnya di fungsi `preprocess_all()`.

### 2.2.2 Matriks Term–Document

Setelah preprocessing, LSA membangun *vocabulary* (himpunan kata unik) dan membuat matriks term–document  $A$  berukuran  $m \times n$ , di mana  $m$  adalah jumlah *unique term* dan  $n$  adalah jumlah dokumen.

Baris-baris matriks  $A$  mewakili kata-kata unik (term), sedangkan kolom-kolomnya mewakili dokumen dalam korpus. Elemen  $a_{ij}$  menunjukkan frekuensi kemunculan kata ke- $i$  dalam dokumen ke- $j$ . Secara matematis, matriks term–document dapat dinyatakan sebagai:

$$A = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m1} & f_{m2} & \cdots & f_{mn} \end{bmatrix},$$

dengan  $f_{ij}$  adalah frekuensi kemunculan kata ke- $i$  dalam dokumen ke- $j$ . Sebagai contoh, misalkan kita memiliki tiga dokumen dengan konten sebagai berikut:

- Dokumen 1: “Panduan membaca buku fiksi secara efektif”
- Dokumen 2: “Buku fiksi dan non-fiksi populer di perpustakaan kampus”
- Dokumen 3: “Cara menata buku di perpustakaan agar mudah dicari”

Maka matriks term-document yang dihasilkan adalah:

Term / Dokumen	Dokumen 1	Dokumen 2	Dokumen 3
panduan	1	0	0
membaca	1	0	0
buku	1	1	1
fiksi	1	1	0
non-fiksi	0	1	0
populer	0	1	0
perpustakaan	0	1	1
menata	0	0	1
mudah	0	0	1
dicari	0	0	1

Kode program dapat ditemukan di `utils.py`.

### 2.2.3 Pembobotan TF-IDF

Matriks term-document  $A$  yang dibangun sebelumnya hanya berisi frekuensi mentah kemunculan kata. Agar representasi lebih informatif, dilakukan pembobotan ulang menggunakan *Term Frequency-Inverse Document Frequency* (TF-IDF), yang memberikan bobot tinggi pada kata yang penting (jarang muncul di seluruh dokumen) dan bobot rendah pada kata yang umum.

Term Frequency (TF) menghitung proporsi kemunculan kata dalam suatu dokumen. Untuk setiap term  $i$  dan dokumen  $j$ :

$$\text{TF}_{ij} = \frac{f_{ij}}{\text{total kata dalam dokumen } j}$$

Inverse Document Frequency (IDF) mengukur seberapa jarang sebuah term muncul dalam keseluruhan korpus:

$$\text{IDF}_i = \log_{10} \left( \frac{N}{1 + df_i} \right)$$

dengan  $N$  jumlah dokumen dan  $df_i$  banyaknya dokumen yang mengandung term  $i$ . Penambahan 1 pada penyebut (*smoothing*) mencegah pembagian dengan nol.

Bobot TF-IDF diperoleh dengan mengalikan keduanya:

$$\text{TF-IDF}_{ij} = \text{TF}_{ij} \times \text{IDF}_i,$$

atau dalam bentuk matriks:

$$\text{TF-IDF} = \text{diag}(\text{IDF}) \cdot \text{TF}.$$

Pada kode program:

- `compute_tf()` menghitung nilai TF per dokumen,
- `compute_idf()` menghitung nilai IDF untuk setiap term,
- `build_matrix()` membangun matriks TF-IDF dengan mengalikan TF dan IDF.

Matriks TF-IDF inilah yang kemudian digunakan sebagai input SVD dalam proses *Latent Semantic Analysis*.

### 2.2.4 Reduksi Dimensi dengan SVD

Analisis semantik laten (latent semantic analysis, LSA) adalah teknik yang digunakan untuk menemukan pola-pola tersembunyi dalam data teks dengan mereduksi dimensi matriks term-document menggunakan singular value decomposition (SVD). Tahap inti LSA adalah melakukan SVD pada matriks TF-IDF:

$$A = U \Sigma V^T.$$

Makna dari masing-masing komponen adalah sebagai berikut:

- $U$  : merepresentasikan hubungan antara *term* (kata) dan *latent topic*,
- $\Sigma$  : berisi *singular values* yang menggambarkan kekuatan atau kontribusi setiap topik,
- $V$  : merepresentasikan hubungan antara dokumen dan *latent topic*.

Untuk mengurangi *noise* dan hanya mempertahankan struktur semantik yang penting, digunakan *truncated SVD* pada  $k$  topik terbesar. Proses reduksi dilakukan dengan mengambil:

$$U_k = U[:, :k], \quad \mathbf{s}_k = \mathbf{s}[:, :k], \quad V_k^T = V^T[:, :k],$$

di mana  $\mathbf{s}$  adalah vektor yang berisi singular values (bukan matriks diagonal).

Kode program diimplementasikan dalam `lsa.py` (fungsi `compute_svd()` dan `reduce_dimension()`), dengan nilai  $k$  default adalah 150.

### 2.2.5 Representasi Dokumen dalam Ruang Semantik Laten

Setiap dokumen direpresentasikan sebagai vektor berdimensi  $k$  dalam ruang topik hasil reduksi SVD. Secara matematis, representasi dokumen ke- $d$  diberikan oleh

$$\mathbf{d}_i = \text{normalize} (\mathbf{s}_k \odot \mathbf{V}_k[:, i]),$$

di mana  $\mathbf{s}_k$  adalah vektor berisi  $k$  singular value terbesar,  $\mathbf{V}_k[:, i]$  adalah kolom ke- $i$  dari matriks  $V_k^T$  (representasi dokumen pada  $k$  topik utama), dan  $\odot$  adalah perkalian elemen per elemen (*element-wise multiplication*). Vektor hasil kemudian dinormalisasi agar memiliki panjang satuan.

Secara intuitif, dokumen dengan pola topik yang mirip akan memiliki vektor yang berdekatan dalam ruang berdimensi rendah ini.

Pada implementasi, proses ini dilakukan di fungsi `compute_doc_vectors()` yang menghitung dan menyimpan vektor dokumen secara *precomputed*. Vektor-vektor ini kemudian digunakan dalam fungsi `search()` dan `recommend_by_id()` untuk menghitung *cosine similarity*.

### 2.2.6 Metode Similaritas

Untuk proses pencarian maupun rekomendasi, digunakan ukuran kemiripan *cosine similarity* yang didefinisikan sebagai

$$\text{sim}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|},$$

di mana  $\mathbf{a} \cdot \mathbf{b}$  adalah hasil perkalian titik (*dot product*) dan  $\|\cdot\|$  menyatakan norma Euclidean.

Metode ini digunakan dalam beberapa konteks:

- Pencarian berbasis dokumen (`search()` dengan input file): pengguna mengunggah file teks, dan sistem mencari dokumen-dokumen yang paling mirip dengan konten file tersebut.
- Rekomendasi dokumen (`recommend_by_id()`): berdasarkan dokumen yang sedang dibaca/dipilih, sistem mencari dokumen lain yang paling mirip menggunakan vektor dokumen yang sudah di-*precompute*.

Pada implementasi, fungsi `cosine_similarity()` digunakan untuk menghitung nilai kemiripan antara vektor query atau vektor dokumen target dengan vektor dokumen lainnya dalam ruang laten hasil SVD.

## 2.3 Teori Singkat dan Jawaban

### 2.3.1 Eigenvalue dan Eigenvector

Eigenvalue dan eigenvector adalah pasangan nilai  $(\lambda, v)$  yang memenuhi persamaan

$$Av = \lambda v,$$

dengan  $A$  adalah matriks persegi,  $v$  adalah eigenvector, dan  $\lambda$  adalah eigenvalue.

**Makna geometris.** Secara geometris, eigenvector merupakan arah yang tidak berubah ketika transformasi linear  $A$  diterapkan, sedangkan eigenvalue menunjukkan faktor skala yang membesar-kecilkan vektor tersebut. Jika  $A$  dianggap sebagai distorsi, maka eigenvector adalah arah yang hanya mengalami penskalaan tanpa rotasi.

**Cara mencari eigenvalue/eigenvector.** Secara prinsip, eigenvalue diperoleh dari persamaan karakteristik

$$\det(A - \lambda I) = 0.$$

Namun metode ini tidak efisien untuk matriks besar. Dalam komputasi modern digunakan algoritma iteratif seperti:

- QR Iteration (mencari semua eigenvalue melalui dekomposisi QR berulang),
- Power Iteration (mendapatkan eigenvalue terbesar),
- QR with shift (mis. Wilkinson shift) untuk konvergensi lebih cepat dan stabil.

Pada tugas besar ini, eigenvalue dihitung menggunakan *QR iteration dengan Wilkinson shift*, dimana dekomposisi QR dilakukan menggunakan metode *Householder reflection*.

### 2.3.2 Singular Value Decomposition

SVD menguraikan sebuah matriks  $A$  menjadi tiga komponen ortogonal:

$$A = U\Sigma V^T,$$

dengan:

- $U$  adalah matriks ortogonal berisi *left singular vectors*,
- $\Sigma$  adalah matriks diagonal berisi *singular values*,
- $V$  adalah matriks ortogonal berisi *right singular vectors*.

**Makna geometris.** Singular values menunjukkan besarnya energi atau variasi pada arah tertentu. Kolom-kolom  $V$  menunjukkan arah fitur dominan yang ditangkap oleh matriks. SVD dapat dianggap sebagai generalisasi dari eigen decomposition untuk matriks yang tidak harus persegi.

### 2.3.3 Jawaban

1. Menurut kami, Singular Value Decomposition (SVD) lebih sesuai untuk digunakan dibandingkan *eigendecomposition* karena *eigendecomposition* hanya dapat diterapkan pada matriks persegi  $n \times n$  dan umumnya mensyaratkan matriks tersebut bersifat simetris. Sementara itu, matriks TF-IDF umumnya berbentuk *rectangular* berukuran  $m \times n$  dan tidak simetris sehingga tidak memenuhi syarat untuk *eigendecomposition*. Selain itu, dengan menggunakan SVD kita dapat melakukan dekomposisi pada matriks apa pun, termasuk matriks besar yang tidak simetris, sehingga metode ini kami nilai lebih fleksibel digunakan untuk melakukan pemrosesan teks dan reduksi dimensi.
2. Karena vektor dan nilai eigen dari matriks covariance menangkap arah variansi terbesar yang ada di dalam data. Matriks covariance menggambarkan setiap fitur seperti setiap pixel pada gambar atau setiap kata yang ada pada dokumen atau teks. Setiap eigenvalue yang menyertai eigenvector akan menunjukkan seberapa besar variansi data yang ada pada arah tersebut. Semakin besar nilai eigenvalue, maka semakin penting pula komponen tersebut dalam sebuah struktur data.
3. Kedua metode tersebut kami rasa kurang optimal terutama ketika gambar kueri memiliki karakteristik yang sangat berbeda dari distribusi gambar pada dataset. Cara kerja dari PCA adalah dengan mengasumsikan bahwa citra yang diuji berada dalam sebuah ruang variansi yang sama dengan citra yang digunakan saat training. Jika gambar input memiliki desain yang sangat berbeda dan resolusi atau komposisi dari warna yang jauh dari rata-rata dataset, maka proyeksinya ke ruang PCA tidak akan akurat, yang mengakibatkan metode ini gagal mengenali kemiripan dari kedua gambar.

## 3 Desain dan Implementasi

### 3.1 Arsitektur Aplikasi (*Root*)

Aplikasi ini merupakan sistem pencarian dan rekomendasi buku berbasis web yang menggunakan arsitektur *client-server*. Aplikasi terdiri dari dua komponen utama: *front end* yang dibangun dengan Next.js dan *back end* yang dibangun dengan Python FastAPI.

Dengan struktur folder keseluruhan di bawah ini:

```
algeo2-ngikut-stress/
├── .gitignore
├── LICENSE
├── README.md
└── data/
    ├── mapper.json
    ├── cache/
    │   ├── vocab.json
    │   ├── tf_list.json
    │   ├── idf.json
    │   ├── book_meta.json
    │   ├── tfidf_matrix.npz
    │   ├── doc_vectors.npy
    │   ├── u_k.npz
    │   └── s_k.npy
    ├── covers/
    ├── covers_grayscale/
    └── txt/
└── docs/
└── src/
    ├── backend/
    └── frontend/
└── test/
```

Dapat dilihat pada struktur folder yang ada di atas, project disusun menjadi beberapa bagian folder utama yaitu `data/`, `docs/`, `src/`, dan `test/`. Folder `data/` menyimpan seluruh data termasuk file `mapper.json`, `cache/` untuk menyimpan hasil preprocessing LSA. Folder `docs/` berisi laporan akhir. Folder `src/` berisi source code aplikasi yang terbagi menjadi `backend/` dan `frontend/`. dan `README.md` berisi dokumentasi cara menjalankan aplikasi.

### 3.2 Arsitektur Aplikasi (*Front End*)

Pada bagian *front end*, kami menggunakan framework Next.js dengan React, TypeScript, Tailwind CSS untuk styling, Shadcn UI, dan ESLint dengan menggunakan npm sebagai package manager.

Dengan struktur folder di bawah ini:

```
frontend/
└── app/
    ├── api/
    │   └── books/
    │       ├── route.ts
    │       └── [id]/
    │           └── route.ts
    ├── search/
    │   └── route.ts
    └── books/
        └── [id]/
            └── page.tsx
    └── search/
        └── [query]/
            └── page.tsx
    └── doc/
```

```
    └── page.tsx
    └── image/
        └── page.tsx
    └── page.tsx
    └── favicon.ico
    └── globals.css
    └── layout.tsx
    └── page.tsx
    └── components/
        └── search/
            ├── doc-search.tsx
            ├── image-search.tsx
            ├── query-result.tsx
            ├── search-bar.tsx
            └── search-result-card.tsx
        └── ui/
            ├── avatar.tsx
            ├── button.tsx
            ├── card.tsx
            ├── dropdown-menu.tsx
            ├── input.tsx
            └── tooltip.tsx
        └── book-card.tsx
        └── footer.tsx
        └── header.tsx
        └── pagination.tsx
    └── lib/
        ├── fonts.ts
        └── utils.ts
    └── fonts/
        └── sf-mono/
            ├── SFMonoRegular.otf
            └── ..
    └── public/
        ├── favicon.ico
        ├── logo-itb.png
        ├── header.jpg
        └── ..
    └── providers/
    └── .gitignore
    └── .next/
    └── components.json
    └── eslint.config.mjs
    └── next.config.ts
    └── node_modules/
    └── package-lock.json
    └── package.json
    └── postcss.config.mjs
    └── README.md
    └── tsconfig.json
```

Dapat dilihat pada struktur folder yang ada diatas, *front end* disusun menjadi beberapa bagian folder seperti public, app, components, dan lib. Folder public berisi file-file assets seperti gambar, ikon, dan file SVG. Folder app mengikuti struktur App Router Next.js dan berisi halaman serta routes dari aplikasi. Folder components berisi React components reusable yang digabung dengan Shadcn UI dan Tailwind CSS. Folder lib berisi utility functions yang digunakan di seluruh aplikasi. Untuk file-file konfigurasi seperti next.config.ts, tailwind.config.ts, tsconfig.json, dan eslint.config.mjs berfungsi untuk mengatur setup framework, styling, type checking, dan code quality dari aplikasi.

### 3.3 Arsitektur Aplikasi (*Back End*)

Pada bagian *back end*, kami menggunakan framework FastAPI dengan Python, NumPy dan SciPy untuk komputasi matriks dan numerik, NLTK untuk tokenizer dalam pemrosesan teks, Pillow untuk pemrosesan gambar, dan Uvicorn sebagai ASGI server.

Dengan struktur folder di bawah ini:

```
backend/
├── main.py
├── requirements.txt
└── app/
    ├── api/
    │   ├── core/
    │   │   ├── config.py
    │   │   └── dependencies.py
    │   └── routers/
    │       ├── books.py
    │       ├── search.py
    │       └── recommend.py
    └── lib/
        ├── books.py
        ├── maths.py
        ├── text/
        │   ├── lsa.py
        │   ├── process.py
        │   └── utils.py
        └── image/
            └── main.py
```

Dapat dilihat pada struktur folder yang ada di atas, *back end* disusun menjadi beberapa bagian folder seperti `app/api` dan `app/lib`. File `main.py` merupakan entry point aplikasi yang menginisialisasi FastAPI. File `requirements.txt` berisi daftar dependency Python yang diperlukan. Folder `app/api/core` berisi konfigurasi aplikasi (`config.py`) dan untuk mengelola singleton instances (`dependencies.py`). Folder `app/api/routers` berisi endpoint handlers yang dipisahkan berdasarkan fitur: `books.py` untuk akses data buku, `search.py` untuk pencarian teks dan gambar, serta `recommend.py` untuk rekomendasi buku. Folder `app/lib` berisi implementasi logika utama, seperti `maths.py`, `books.py` untuk akses data, `text/` untuk pemrosesan teks dan model LSA, dan `image/` untuk pemrosesan gambar dan PCA.

#### 3.3.1 Kelas-Kelas Penting

**Kelas Settings** Kelas ini mengelola konfigurasi aplikasi, yaitu path directory data dan parameter LSA.

**Kelas LSA\_Model** Kelas untuk sistem pencarian berbasis *Latent Semantic Analysis*. Atribut penting meliputi:

- `docs`: Token per dokumen hasil preprocessing
- `vocab`: Vocabulary unik dari seluruh kumpulan teks
- `tf_list`, `idf`: Term Frequency dan Inverse Document Frequency
- `matrix`: Matriks TF-IDF
- `U_k`, `S_k`, `VT_k`: Hasil dekomposisi SVD yang sudah direduksi
- `doc_vectors`: Vektor dokumen di ruang LSA

Method penting:

- `preprocess_all()`: Tokenisasi, normalisasi, stopword removal, stemming
- `build_matrix()`: Membangun matriks TF-IDF sparse
- `compute_svd()`: Dekomposisi SVD menggunakan QR iteration dengan Householder reflection dan Wilkinson shift

- `query_to_vector(query, k)`: Proyeksi query ke ruang LSA
- `search(query, top_k, k)`: Pencarian dokumen dengan cosine similarity
- `recommend_by_id(book_id, top_k)`: Rekomendasi berdasarkan kemiripan dokumen

**Kelas ImageProcessor** Menangani pencarian gambar menggunakan metode *Eigencovers* (adaptasi dari Eigenfaces). Atribut penting:

- `eigencovers`: Matriks eigenvector dari SVD
- `avg_vector`: Rata-rata vektor gambar untuk centering
- `image_coefficients_normalized`: Koefisien ternormalisasi setiap gambar

Method penting:

- `_prepare_and_process_images()`: Konversi grayscale dan SVD
- `search_cover_by_image()`: Pencarian cover berdasarkan similarity

**Kelas TextProcessor** Memuat dan memproses file teks dari `mapper.json`.

**Kelas BookRepository** Repository pattern untuk akses metadata buku dari file `mapper.json`.

### Fungsi Matematika (maths.py)

- `svd(A)`: Implementasi SVD berbasis eigen-decomposition
- `householder_qr(A)`: Dekomposisi QR dengan Householder reflection
- `qr_eigen_shift(A)`: QR iteration dengan Wilkinson shift

### Fungsi Utility Teks (utils.py)

- `preprocess_text(text)`: Pipeline preprocessing teks
- `compute_tf(tokens)`: Menghitung Term Frequency
- `compute_idf(docs)`: Menghitung Inverse Document Frequency
- `cosine_similarity(v1, v2)`: Menghitung cosine similarity

## 3.4 Spesifikasi Bonus

### 3.4.1 Kontainerisasi dengan Docker dan Deployment

Aplikasi dikemas menggunakan Docker untuk memastikan konsistensi lingkungan pengembangan dan produksi. Arsitektur kontainer terdiri dari tiga layanan utama yang dikelola menggunakan Docker Compose.

#### Arsitektur Kontainer

1. **Data Downloader**: Kontainer sementara yang bertugas mengunduh dan mengekstrak dataset dari Kaggle secara otomatis jika belum tersedia.
2. **Backend**: Kontainer Python yang menjalankan API server menggunakan FastAPI dan Uvicorn pada port 8000.
3. **Frontend**: Kontainer Node.js yang menjalankan aplikasi Next.js pada port 3000.

**Deployment** Aplikasi dapat di-deploy dengan satu perintah: `docker compose up -build`

### 3.4.2 Beautiful UI/UX

**Responsive Design** Implementasi desain pada website yang kami kembangkan menggunakan Tailwind CSS dengan style yang cocok dengan layar mobile dan layar tablet/desktop. Hal ini dilakukan menggunakan breakpoints (`sm:`, `md:`, `lg:`). Contoh implementasi pada grid buku: `grid-cols-2 sm:grid-cols-3 md:grid-cols-4 lg:grid-cols-5`.

**Visual Hierarchy** Penerapan hierarki visual yang jelas untuk mengarahkan perhatian pengguna melalui: typography scaling, kontras warna, dan badge informatif (id buku, score)

**Consistency** Konsistensi desain di seluruh aplikasi dicapai melalui penggunaan Tailwind yang memudahkan pewarnaan yang konsisten dan komponen yang reusable.

**Visual Feedback** Aplikasi memberikan feedback visual pada interaksi, misalnya: animasi hover state, loading animation, tooltip, highlighting pagination

**Drag and Drop Interface** Fitur upload file yang *user-friendly* dengan dukungan drag-and-drop.

### 3.4.3 Search by Document

Aplikasi web memiliki fitur pencarian buku berdasarkan dokumen teks yang diunggah oleh pengguna. Input dapat berupa sebuah file `.txt`. Output pencarian menampilkan buku-buku yang paling relevan dengan dokumen yang diunggah berdasarkan algoritma *Latent Semantic Analysis* (LSA).

Berikut adalah alur kerja fitur pencarian berdasarkan dokumen:

1. Pengguna mengunggah file dokumen berformat `.txt` melalui antarmuka web
2. Sistem membaca konten teks dari file yang diunggah
3. Dokumen query diproses menggunakan pipeline TF-IDF yang sama dengan dokumen dalam database
4. Vektor TF-IDF dokumen query diproyeksikan ke ruang semantik LSA menggunakan matriks  $U_k$  dan  $S_k$
5. Sistem menghitung *cosine similarity* antara vektor query dengan seluruh vektor dokumen dalam database
6. Hasil pencarian diurutkan berdasarkan skor kemiripan tertinggi dan ditampilkan kepada pengguna

**Implementasi Backend** Pada sisi backend, proses pencarian dokumen diimplementasikan dengan langkah-langkah berikut:

**Preprocessing Dokumen** Kurang-lebih sama seperti preprocessing dokumen data, yaitu: normalisasi, tokenization, stopwords removal, stemming.

**Perhitungan TF-IDF** Vektor TF-IDF untuk dokumen query dihitung menggunakan formula:

$$\text{TF-IDF}(t, d_q) = \text{TF}(t, d_q) \times \text{IDF}(t) \quad (1)$$

di mana  $\text{TF}(t, d_q)$  adalah frekuensi term  $t$  dalam dokumen query  $d_q$ , dan  $\text{IDF}(t)$  adalah nilai *Inverse Document Frequency* yang telah dihitung sebelumnya dari korpus.

**Proyeksi ke Ruang LSA** Vektor TF-IDF query  $\mathbf{q}$  diproyeksikan ke ruang semantik berdimensi  $k$ :

$$\hat{\mathbf{q}}_k = \frac{U_k^T \mathbf{q}}{\|U_k^T \mathbf{q}\|} \quad (2)$$

Vektor dokumen dalam ruang LSA dihitung sebagai:

$$\hat{\mathbf{d}}_i = \frac{S_k \odot V_k^{(i)}}{\|S_k \odot V_k^{(i)}\|} \quad (3)$$

di mana  $\odot$  adalah *element-wise multiplication*,  $U_k \in R^{m \times k}$  adalah matriks singular kiri,  $S_k \in R^k$  adalah vektor nilai singular, dan  $V_k^{(i)}$  adalah kolom ke- $i$  dari matriks singular kanan.

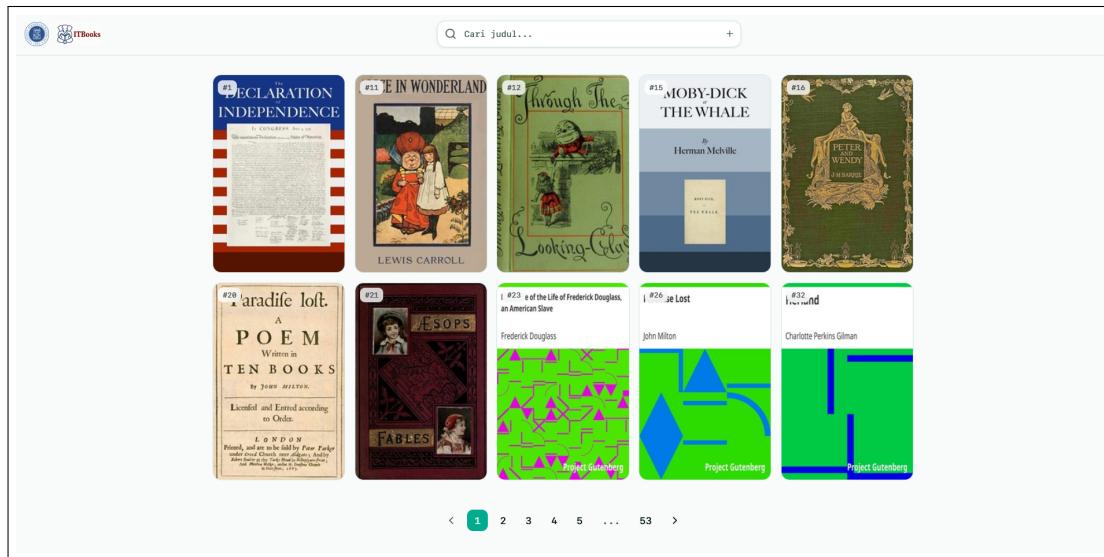
**Perhitungan Similarity** Kemiripan dihitung menggunakan *cosine similarity*:

$$\text{sim}(\mathbf{q}, \mathbf{d}_i) = \hat{\mathbf{q}}_k \cdot \hat{\mathbf{d}}_i \quad (4)$$

Karena kedua vektor sudah dinormalisasi, *cosine similarity* menjadi *dot product* biasa.

## 4 Eksperimen

### 4.1 Eksperimen pada Laman Utama



Homepage

### 4.2 Eksperimen pada Laman Detail Buku

The screenshot shows the detail page for the book "Crime and Punishment" by Fyodor Dostoevsky. At the top, there is a search bar with the placeholder "Cari judul..." and a plus sign button. To the left of the search bar is a book cover thumbnail for "Crime and Punishment". To the right of the search bar is a large content area.

**Content**

CRIME AND PUNISHMENT

CRIME AND PUNISHMENT \*\*\*

CRIME AND PUNISHMENT

By Fyodor Dostoevsky

Translated By Constance Garnett

TRANSLATOR'S PREFACE

A few words about Dostoevsky himself may help the English reader to understand his work.

**Recommended Books**

1 #13415 The Dog and the Bone 66.0%	1 #13437 Short Stories 29.0%	1 #36834 ; and Other 29.0%	1 #680 In the Underworld 73.0%	1 #1756 Scores from Crime 16.0%
Aleko Padouch Chakhot	Thomas Seltzer	Fyodor Dostoevsky	Fyodor Dostoevsky	Aleko Padouch Chakhot

Laman detail buku

### 4.3 Eksperimen Pencarian Menggunakan Judul

The screenshot shows the search results for the title "jungle". The search bar at the top contains the text "Cari judul...". Below it, the heading "Hasil Pencarian untuk 'jungle'" is displayed, followed by the text "Ditemukan 2 buku". Two book covers are shown side-by-side:

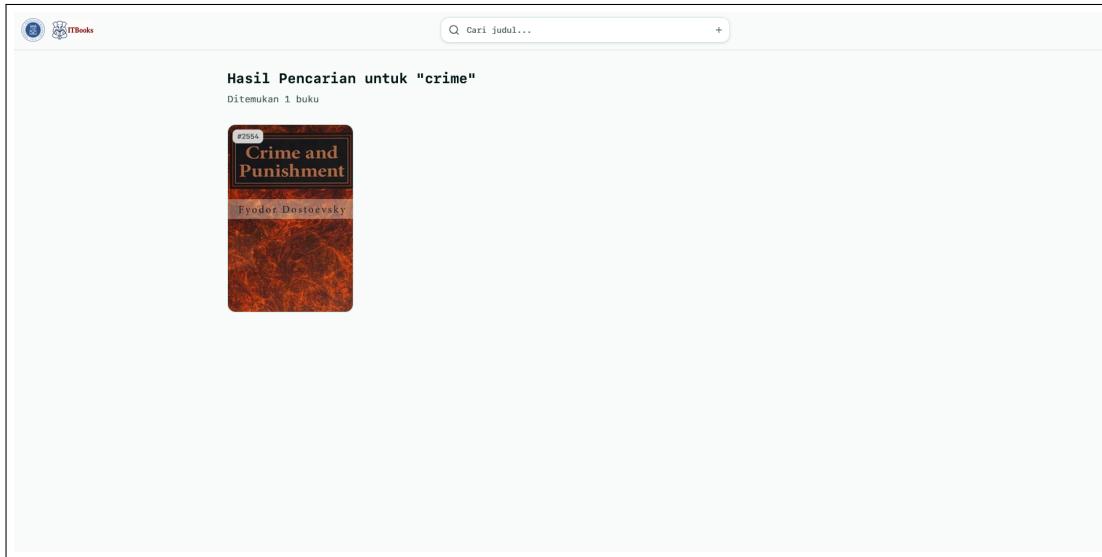
- THE JUNGLE** by UPTON SINCLAIR (Cover #2148)
- THE JUNGLE BOOK** by RUDYARD KIPLING (Cover #236)

Pencarian menggunakan judul 1

The screenshot shows the search results for the title "sherlock". The search bar at the top contains the text "Cari judul...". Below it, the heading "Hasil Pencarian untuk 'sherlock'" is displayed, followed by the text "Ditemukan 3 buku". Three book covers are shown side-by-side:

- THE RETURN OF Sherlock Holmes** by A CONAN DOYLE (Cover #188)
- MEMOIRS OF Sherlock Holmes** by A CONAN DOYLE (Cover #834)
- ADVENTURES OF Sherlock Holmes** by A CONAN DOYLE (Cover #1661)

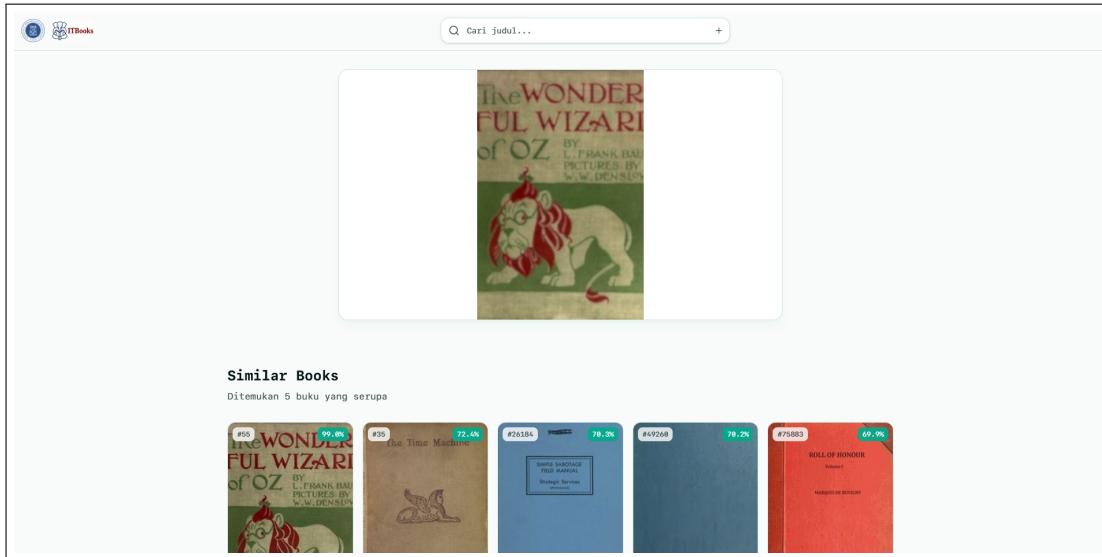
Pencarian menggunakan judul 2



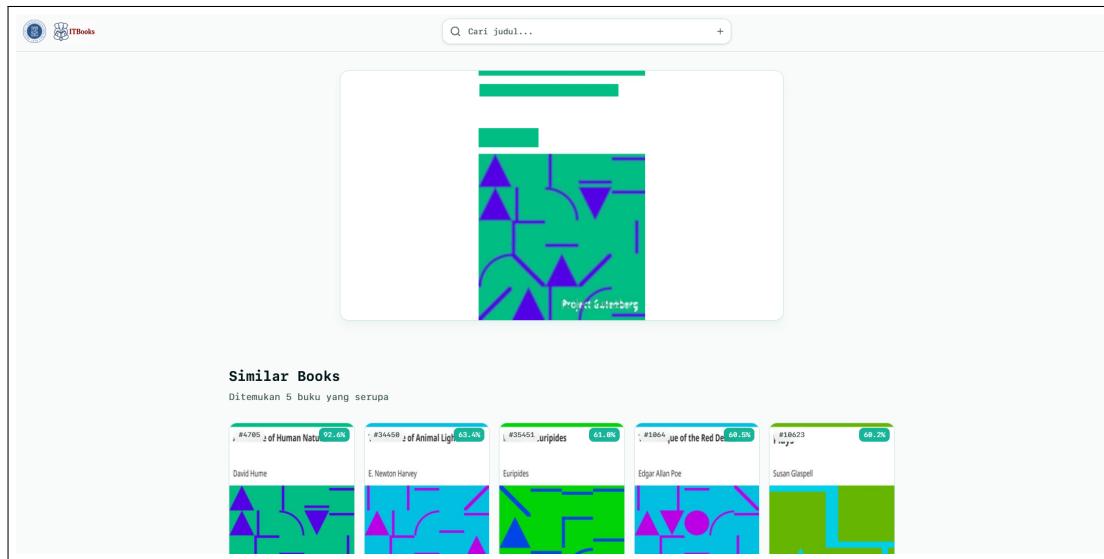
Pencarian menggunakan judul 3

Pencarian menggunakan judul sudah benar untuk 3 test case diatas. Pencarian menggunakan substring matching.

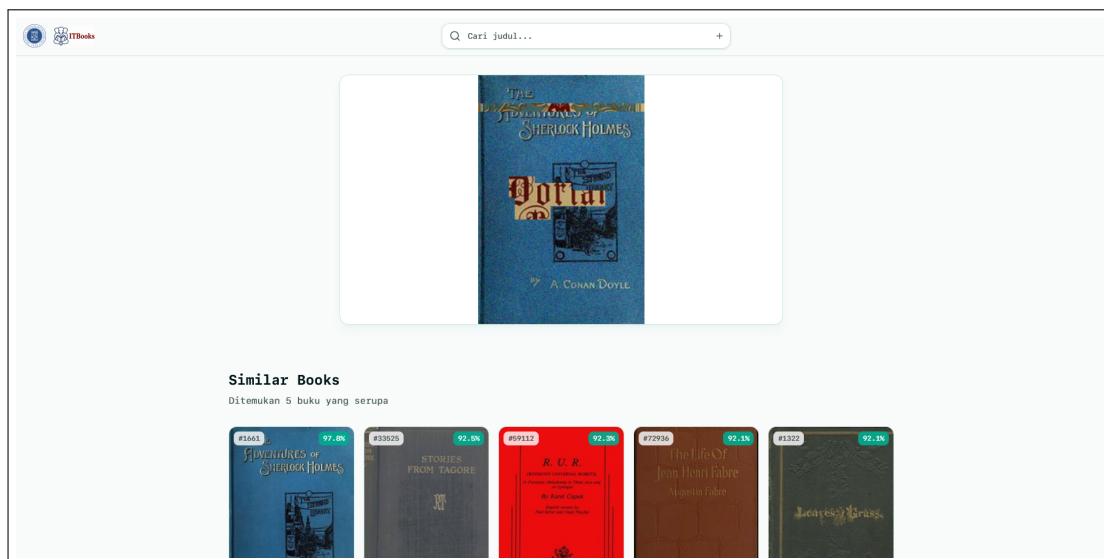
#### 4.4 Eksperimen Pencarian Menggunakan Upload Image (dari asisten)



Pencarian menggunakan gambar 1



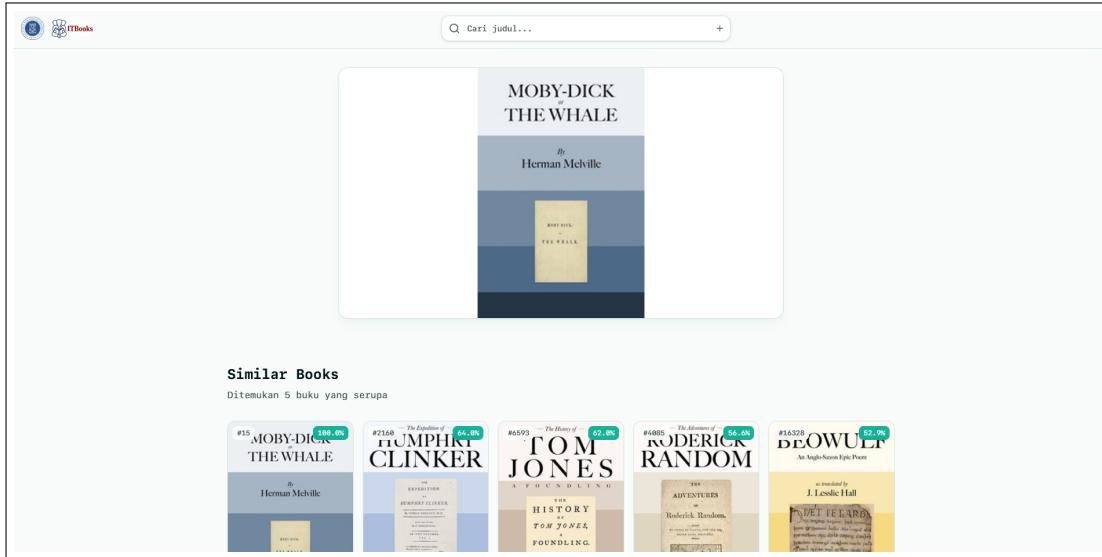
Pencarian menggunakan gambar 2



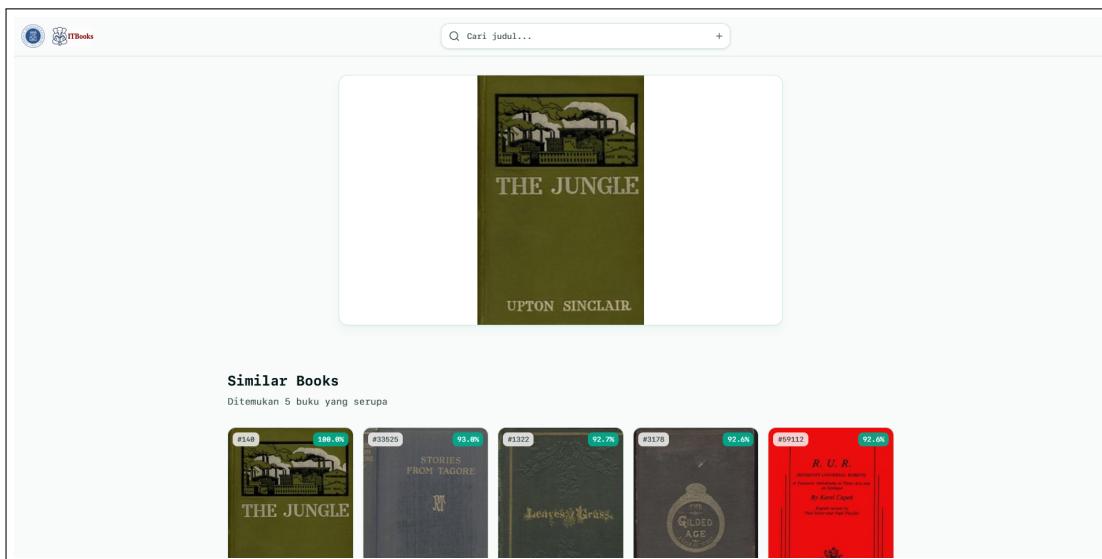
Pencarian menggunakan gambar 3

Pencarian menggunakan cover image sudah benar untuk 3 test case yang diberikan oleh asisten. Hasil pencarian memberikan score akurasi/confidence yang lumayan tinggi.

#### 4.5 Eksperimen Pencarian Menggunakan Upload Image



Pencarian menggunakan gambar 1



Pencarian menggunakan gambar 2

Pencarian menggunakan cover image sudah benar untuk 2 test case random. Hasil pencarian memberikan score akurasi/confidence yang lumayan tinggi.

## 4.6 Eksperimen Rekomendasi

The screenshot shows the IfBooks platform interface. At the top, there is a search bar with the placeholder "Cari judul...". Below the search bar, the book cover for "The Return of Sherlock Holmes" by Sir Arthur Conan Doyle is displayed. To the right of the book cover, the author's name is listed, followed by a "Contents" section which lists several chapters. Below this, a "Recommended Books" section is shown with five book covers: "MEMOIRS of SHERLOCK HOLMES", "ADVENTURES of SHERLOCK HOLMES", "SIGN of the BASKERVILLES", "SIGN of the BASKERVILLES" (repeated), and "Scarlet". At the bottom of the page, there is a footer with the text "tugas besar algeo" and "ngikut-stress-team © 2025".

Rekomendasi 1

The screenshot shows the IfBooks platform interface. At the top, there is a search bar with the placeholder "Pari judul...". Below the search bar, the book cover for "A Treatise of Human Nature" by David Hume is displayed. To the right of the book cover, the author's name is listed, followed by a "CONTENTS" section which includes "VOLUME I", "INTRODUCTION BY THE AUTHOR.", and "BOOK I OF THE UNDERSTANDING". Below this, a "Recommended Books" section is shown with five book covers: "A Treatise of Human Nature", "HUMAN Understanding", "on the Method of Nature", "Conducting One's Reason and of Sensing", and "of Aristotle". At the bottom of the page, there is a footer with the text "tugas besar algeo" and "ngikut-stress-team © 2025".

Rekomendasi 2

Rekomendasi untuk buku sudah benar untuk 2 test case yang diberikan oleh asisten. Hasil rekomendasi memberikan score kedekatan yang lumayan tinggi. Contohnya, untuk buku 1, rekomendasi lainnya adalah buku Sherlock Holmes.

## 4.7 Eksperimen Pencarian Menggunakan Upload Document (bonus)

The screenshot shows the ITBooks search interface. At the top, there is a search bar with the placeholder "Cari judul...". Below the search bar, a preview window displays the content of the uploaded file "1.txt". The content is titled "THE DECLARATION OF INDEPENDENCE OF THE UNITED STATES OF AMERICA \*\*\*" and includes a detailed description of its history and storage details. Below the preview, a section titled "Similar Books" shows five recommended books with their titles, authors, and covers.

Pencarian dokumen 1

The screenshot shows the ITBooks search interface. At the top, there is a search bar with the placeholder "Cari judul...". Below the search bar, a preview window displays the content of the uploaded file "11.txt". The content is titled "ALICE'S ADVENTURES IN WONDERLAND \*\*\*" and includes a note about it being an illustration. Below the preview, a section titled "Similar Books" shows five recommended books with their titles, authors, and covers.

Pencarian dokumen 2

Pencarian dokumen 3 (isi txtnya memang Moby Dick)

Pencarian menggunakan dokumen sudah benar untuk 2 test case random. Hasil pencarian memberikan score kedekatan yang lumayan tinggi. Untuk test case 1, txt dari buku #1 dilakukan search dan mendapatkan hasil yang persis. Untuk test case 2, diberikan potongan paragraf dari Moby Dick dan diberi judul "the jungle book", tapi hasil pencarian tetap menghasilkan buku Moby Dick.

## 5 Penutup

### 5.1 Kesimpulan

Pada proyek tugas besar ini, kami telah berhasil membuat *EigenPustaka* yang merupakan sebuah aplikasi yang dapat memungkinkan pengguna untuk mencari buku dengan sistem pencarian dan rekomendasi buku yang berbasis *Latent Semantic Analysis* (LSA) dan *image retrieval* yang berbasis *Principal Component Analysis* (PCA). Sistem pada aplikasi yang telah kami bangun, mampu untuk melakukan preprocessing teks, membangun matriks TF-IDF, dan menerapkan SVD untuk reduksi dimensi, serta menghasilkan vektor yang merepresentasikan dokumen yang dapat digunakan untuk melakukan *semantic search* dan menampilkan rekomendasi buku yang akurat. Fitur utama dari aplikasi yang telah kami bangun mencakup pencarian berbasis dokumen, rekomendasi berdasarkan dokumen yang serupa, dan temu balik gambar.

### 5.2 Saran

**Nathan** Deadlinenya nambah sebulan dong banh :D

**Azri** setuju banget sama Nathan

### 5.3 Refleksi

**Nathan** Seru sih tapi gamau lagi :3

**Azri** Aku setuju sama Nathan

**Rasyad** Foto ini aja



## Daftar Pustaka

- [1] D. Bindel, Lecture 18: Eigenvalues and Singular Values, Cornell University, 2009. [Online]. Available: <https://www.cs.cornell.edu/~bindel/class/cs6210-f09/lec18.pdf>
- [2] 3Blue1Brown, Essence of Linear Algebra: Singular Value Decomposition, YouTube, 2016. [Online]. Available: [https://www.youtube.com/watch?v=H3DNv\\_7XVhU](https://www.youtube.com/watch?v=H3DNv_7XVhU)
- [3] Rinaldi Munir, IF2123 Aljabar Geometri - Semester I Tahun 2025/2026, 2025. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2025-2026/algeo25-26.htm>
- [4] Olney, Andrew McGregor, Large-scale latent semantic analysis, 2011. [Online]. Available: <https://link.springer.com/content/pdf/10.3758/s13428-010-0050-z.pdf>

## 6 Lampiran

**Link Repository Github** <https://github.com/IRK-23/algeo2-ngikut-stress>

**Link Deployment** <http://algeo.ptpb24.xyz/>