

LAPORAN TUGAS BESAR

IF2123 Aljabar Linear Geometri

Content Based Image Retrieval (CBIR)



Dipersiapkan oleh:

SmiLens

Maulvi Ziadinda M (13522122)

M. Fauzan Azhim (13522153)

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

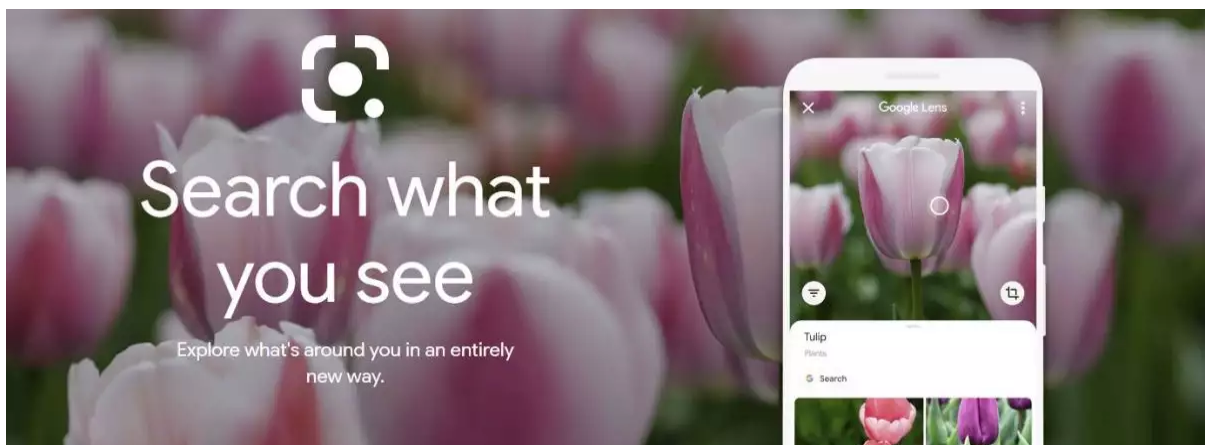
DAFTAR ISI

BAB I.....	3
BAB II.....	4
A. CBIR.....	4
1. CBIR dengan parameter warna.....	4
2. CBIR dengan parameter tekstur.....	6
B. Pengembangan Web.....	8
1. Website.....	8
2. ReactJS.....	8
3. FastApi.....	9
BAB III.....	10
A. Langkah Penyelesaian Masalah.....	10
B. Pemetaan Masalah pada Aljabar Geometri.....	11
1. Pemetaan Masalah pada CBIR Warna.....	11
2. Pemetaan Masalah pada CBIR Tekstur.....	11
3. Integrasi dengan Aljabar Geometri pada Front End dan Back End.....	11
C. Ilustrasi Kasus dan Penyelesaian.....	12
BAB IV.....	14
A. Implementasi Program Utama.....	14
1. CBIR dengan Parameter Warna.....	14
2. CBIR dengan Parameter Tekstur.....	14
B. Struktur Program.....	18
1. Front End.....	18
2. Backend.....	20
C. Penggunaan Program.....	20
1. Mencari Gambar Menggunakan Gambar Lain.....	20
2. Mencari Gambar Menggunakan Kamera.....	21
D. Hasil Pengujian.....	21
E. Analisis Desain Solusi.....	21
BAB V.....	23
A. Kesimpulan.....	23
B. Saran.....	23
C. Komentar.....	23
D. Refleksi.....	23
DAFTAR PUSTAKA.....	24

BAB I

DESKRIPSI MASALAH

Dalam era digital, jumlah gambar yang dihasilkan dan disimpan semakin meningkat dengan pesat, baik dalam konteks pribadi maupun profesional. Peningkatan ini mencakup berbagai jenis gambar, mulai dari foto pribadi, gambar medis, ilustrasi ilmiah, hingga gambar komersial. Terlepas dari keragaman sumber dan jenis gambar ini, sistem temu balik gambar (*image retrieval system*) menjadi sangat relevan dan penting dalam menghadapi tantangan ini. Dengan bantuan sistem temu balik gambar, pengguna dapat dengan mudah mencari, mengakses, dan mengelola koleksi gambar mereka. Sistem ini memungkinkan pengguna untuk menjelajahi informasi visual yang tersimpan di berbagai platform, baik itu dalam bentuk pencarian gambar pribadi, analisis gambar medis untuk diagnosis, pencarian ilustrasi ilmiah, hingga pencarian produk berdasarkan gambar komersial. Salah satu contoh penerapan sistem temu balik gambar yang mungkin kalian tahu adalah Google Lens.



Gambar 1. Contoh penerapan *information retrieval system* (Google Lens)

Di dalam Tugas Besar 2 ini, akan diimplementasikan sistem temu balik gambar yang sudah dijelaskan sebelumnya dengan memanfaatkan Aljabar Vektor dalam bentuk sebuah *website*, dimana hal ini merupakan pendekatan yang penting dalam dunia pemrosesan data dan pencarian informasi. Dalam konteks ini, aljabar vektor digunakan untuk menggambarkan dan menganalisis data menggunakan pendekatan klasifikasi berbasis konten (*Content-Based Image Retrieval* atau CBIR), di mana sistem temu balik gambar bekerja dengan mengidentifikasi gambar berdasarkan konten visualnya, seperti warna dan tekstur.

BAB II

LANDASAN TEORI

A. CBIR

Content-Based Image Retrieval (CBIR) adalah sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan kontennya. Proses ini dimulai dengan ekstraksi fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk. Setelah fitur-fitur tersebut diekstraksi, mereka diwakili dalam bentuk vektor atau deskripsi numerik yang dapat dibandingkan dengan gambar lain. Kemudian, CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan ini digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling mirip dengan gambar yang dicari. Proses CBIR membantu pengguna dalam mengakses dan mengeksplorasi koleksi gambar dengan cara yang lebih efisien, karena tidak memerlukan pencarian berdasarkan teks atau kata kunci, melainkan berdasarkan kesamaan nilai citra visual antara gambar-gambar tersebut. CBIR diterapkan dengan dua parameter yaitu sebagai berikut.

1. CBIR dengan parameter warna

Pada CBIR ini *input* dari sebuah *image* akan dibandingkan dengan *image* yang dimiliki oleh dataset, hal ini dilakukan dengan cara mengubah *image* yang berbentuk RGB menjadi sebuah metode histogram warna yang lebih umum.

Histogram warna adalah frekuensi dari berbagai warna yang ada pada ruang warna tertentu hal ini dilakukan untuk melakukan pendistribusian warna dari *image*. Histogram warna tidak bisa mendeteksi sebuah objek yang spesifik yang terdapat pada *image* dan tidak bisa mendeskripsikan posisi dari warna yang didistribusikan.

Pembentukan ruang warna perlu dilakukan dalam rangka pembagian nilai citra menjadi beberapa *range* yang lebih kecil. Hal itu dilakukan untuk membuat sebuah histogram warna yang setiap interval tiap *range* dianggap sebagai *bin*. Histogram warna dapat dihitung dengan menghitung piksel yang menyatakan nilai warna pada setiap interval. Fitur warna mencakup histogram warna global dan histogram warna blok.

Pada perhitungan histogram, warna global HSV lebih dipilih karena warna tersebut dapat digunakan pada kertas (*background* berwarna putih) yang lebih umum untuk digunakan. Maka dari itu, perlu dilakukan konversi warna dari RGB ke HSV dengan prosedur sebagai berikut.

- a. Nilai dari RGB harus dinormalisasi dengan mengubah nilai *range* [0, 255] menjadi [0, 1]

$$R' = \frac{R}{255} \quad G' = \frac{G}{255} \quad B' = \frac{B}{255}$$

b. Mencari C_{max} , C_{min} , dan Δ

$$C_{max} = \max(R', G', B')$$

$$C_{min} = \min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

c. Selanjutnya gunakan hasil perhitungan di atas untuk mendapatkan nilai HSV

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right) & , C' \text{ max} = R' \\ 60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right) & , C' \text{ max} = G' \\ 60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right) & , C' \text{ max} = B' \end{cases}$$

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

$$V = C_{maks}$$

Setelah mendapatkan nilai HSV lakukanlah perbandingan antara *image* dari input dengan dataset dengan menggunakan *cosine similarity*

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

Dengan A dan B adalah vektor dan n adalah jumlah dimensi dari vektor. Tingkat kemiripan dihitung dari seberapa besar hasil dari *Cosine Similarity*.

Untuk melakukan pencarian histogram, blok *image* dibagi menjadi $n \times n$ blok. Setiap blok akan menjadi tidak terlalu signifikan jika blok-blok tersebut terlalu besar dan akan meningkatkan waktu dalam memprosesnya jika ukuran dari blok terlalu kecil. Pada pencarian blok ini agar lebih efektif akan digunakan 3×3 blok. Nilai representatif dari sebuah blok akan dinyatakan dengan melakukan kalkulasi nilai rata-rata HSV dari blok terkait.

2. CBIR dengan parameter tekstur

CBIR dengan perbandingan tekstur dilakukan menggunakan suatu matriks yang dinamakan *co-occurrence matrix*. Matriks ini digunakan karena dapat melakukan pemrosesan yang lebih mudah dan cepat. Vektor yang dihasilkan juga mempunyai ukuran yang lebih kecil. Misalkan terdapat suatu gambar I dengan $n \times m$

piksel dan suatu parameter offset $(\Delta x, \Delta y)$, Maka dapat dirumuskan matriksnya sebagai berikut:

Dengan menggunakan nilai i dan j sebagai nilai intensitas dari gambar dan p serta q sebagai posisi dari gambar, maka *offset* Δx dan Δy bergantung pada arah θ dan jarak yang digunakan melalui persamaan di bawah ini dengan nilai θ adalah 0° , 45° , 90° , dan 135° .

$$C_{\Delta x, \Delta y}(i, j) = \sum_{p=1}^n \sum_{q=1}^m \begin{cases} 1, & \text{jika } I(p, q) = i \text{ dan } I(p + \Delta x, q + \Delta y) = j \\ 0, & \text{jika lainnya} \end{cases}$$

Setelah didapat *co-occurrence matrix* akan *symmetric matrix* dengan menjumlahkan *co-occurrence matrix* dengan hasil transpose-nya. Lalu cari *matrix normalization* dengan persamaan.

$$\text{MatrixNorm} = \frac{\text{MatrixOcc}}{\sum \text{MatrixOcc}}$$

Langkah-langkah dalam CBIR dengan parameter tekstur adalah sebagai berikut :

- Konversi warna gambar menjadi *grayscale*, ini dilakukan karena warna tidaklah penting dalam penentuan tekstur. Oleh karena itu, warna RGB dapat diubah menjadi suatu warna *grayscale* Y dengan rumus:

$$Y = 0.29 \times R + 0.587 \times G + 0.114 \times B$$

- Lakukan kuantifikasi dari nilai *grayscale*. Karena citra *grayscale* berukuran 256 piksel, maka matriks yang berkoresponden akan berukuran 256×256 . Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.
- Dari *co-occurrence matrix* bisa diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity*. Persamaan yang dapat digunakan untuk mendapatkan nilai 3 komponen tersebut antara lain :

Contrast:

$$\sum_{i,j=0}^{\text{dimensi} - 1} P_{i,j} (i - j)^2$$

Homogeneity :

$$\sum_{i,j=0}^{\text{dimensi} - 1} \frac{P_{i,j}}{1 + (i - j)^2}$$

Entropy :

$$-\left(\sum_{i,j=0}^{\text{dimensi}-1} P_{i,j} \times \log P_{i,j}\right)$$

Keterangan : P merupakan matriks *co-occurrence*

Dari ketiga komponen tersebut, dibuatlah sebuah vektor yang akan digunakan dalam proses pengukuran tingkat kemiripan.

- d. Ukurlah kemiripan dari kedua gambar dengan menggunakan Teorema *Cosine Similarity*, yaitu:

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

Disini A dan B adalah dua vektor dari dua gambar. Semakin besar hasil *Cosine Similarity* kedua vektor maka tingkat kemiripannya semakin tinggi.

B. Pengembangan Web

Website atau situs juga dapat diartikan sebagai kumpulan halaman yang menampilkan informasi data teks, data gambar, data animasi, suara, video, dan gabungan dari semuanya, baik yang bersifat statis maupun dinamis yang membentuk satu rangkaian bangunan yang saling terkait dimana masing-masing dihubungkan dengan jaringan-jaringan halaman atau hyperlink (Dewi, Fauriatun, &, Nurul Rahmadan, 2021).

1. Website

Website memiliki berbagai unsur, namun unsur website yang penting dan utama adalah desain. Desain website menentukan kualitas dan keindahan sebuah website. Desain sangat berpengaruh kepada penilaian user akan bagus tidaknya sebuah website (Dewi, Fauriatun, &, Nurul Rahmadan, 2021). Dengan desain yang baik, website akan semakin mudah untuk bekerja dengan efisien, efektif, dan tentu nyaman digunakan oleh user. Website juga memiliki banyak manfaat. Manfaat website saat ini, tidak hanya terbatas di bidang IT saja. Mengembangkan sebuah bisnis dengan memanfaatkan website sangat diperlukan di masa globalisasi saat ini. Keberadaan website saat ini banyak sekali ditemukan seiring jumlah permintaan yang terus meningkat. (Dewi, Fauriatun, &, Nurul Rahmadan, 2021).

2. ReactJS

React, biasa dikenal sebagai React atau React.Js, merupakan sebuah library JavaScript dengan sifat open source yang digunakan untuk mempermudah dalam membangun tampilan antarmuka pengguna (UI) menjadi lebih responsive dan

interaktif. React biasa digunakan untuk mengembangkan tampilan pada Single Page Application (SPA) dan mobile application (TungKhuat, 2018).

React diciptakan oleh seorang insinyur perangkat lunak dari Facebook yang bernama Jordan Walke pada tahun 2011 dan secara resmi digunakan oleh Facebook pada bagian news feed mereka. Selain itu, Instagram juga memilih React untuk digunakan pada sistem mereka. Sejak saat itu, React telah berkembang menjadi salah satu library JavaScript yang banyak digunakan hingga saat ini (Anup Satyal, 2020).

React hanya merepresentasikan bagian view dari konsep MVC (Model-View-Controller) (Archana Bhalla, 2020). Artinya, React hanya digunakan untuk membuat tampilan antarmukanya saja tanpa memiliki kemampuan untuk berhubungan dengan basis data.

React.js memiliki aturan tersendiri dalam menuliskan kode pada setiap komponennya, yaitu dengan menggunakan JSX yang memungkinkan developer untuk menuliskan sintaks HTML di dalam JavaScript. Dengan mengkombinasikan dua hal tersebut antara HTML dan JavaScript, penulisan kode menjadi lebih sederhana dan ringkas dalam pendeskripsian tampilan antarmuka. JSX sendiri merupakan ekstensi sintaks untuk ECMAScript tanpa ada definisi semantik tertentu. Pada dasarnya React.js bisa digunakan tanpa JSX, namun tanpa JSX penulisan kode menjadi lebih panjang dan sulit. Sedangkan dengan menggunakan JSX dapat memudahkan pengembang dalam penulisan kode dan membantu saat melakukan error debugging (Tung Khuat, 2018).

3. FastApi

Application programming interface (API) adalah suatu dokumentasi yang terdiri dari antarmuka, fungsi, kelas, struktur dan sebagainya untuk membuat sebuah perangkat lunak (Ramadhani, 2015). Dengan adanya API ini, maka memudahkan programmer untuk “membongkar” suatu software, kemudian dapat dikembangkan atau diintegrasikan dengan perangkat lunak yang lain. API dapat dikatakan sebagai koneksi antar aplikasi dengan aplikasi lain yang memungkinkan seorang programmer menggunakan sistem function (Pranata, Hijriani, & Junaidi, 2018). Proses ini dikelola melalui sistem operasi. Keunggulan dari menggunakan API adalah memungkinkan suatu aplikasi dengan aplikasi lainnya dapat saling berkomunikasi dan berinteraksi.

Pada tahun 2018, Sebastian Ramirez mengembangkan kerangka kerja web Python yang disebut FastAPI. Menurut pendapatnya, FastAPI seharusnya menjadi kerangka kerja terbaik untuk mengembangkan layanan REST API. Framework FastApi dijalankan secara asynchronous dan mudah diintegrasikan dengan OpenAPI-schema, yang membuatnya lebih mudah untuk bekerja dengan Swagger dan ReDoc (Kornienko, Mishina, & Melnikov, 2021). Meskipun FastApi belum lama dikembangkan, FastAPI memantapkan dirinya sebagai tools yang berkualitas. Banyak perusahaan besar mulai menggunakan untuk mengembangkan API mereka.

BAB III

ANALISIS PEMECAHAN MASALAH

A. Langkah Penyelesaian Masalah

Dalam tugas besar ini, penyelesaian masalah kami difokuskan pada implementasi Content-Based Image Retrieval (CBIR) dengan memanfaatkan komponen warna atau tekstur. Penerapan CBIR ini dilakukan melalui pengembangan sebuah aplikasi web menggunakan ReactJS sebagai framework front end dan FastAPI sebagai back end. Adapun langkah-langkah penyelesaian masalah dapat diuraikan sebagai berikut.

Pertama, kami melakukan analisis dan pemilihan dataset yang sesuai dengan kebutuhan CBIR. Dataset yang digunakan harus mencakup variasi gambar dengan komponen warna dan tekstur yang bervariasi untuk memastikan keberagaman hasil pencarian. Selanjutnya, kami melakukan pra-pemrosesan data pada setiap citra untuk mengekstrak informasi yang relevan terkait warna dan tekstur.

Kedua, dalam pengembangan front end menggunakan ReactJS, kami merancang antarmuka pengguna yang intuitif untuk memudahkan pengguna dalam melakukan pencarian gambar. Kami menambahkan fitur pencarian berdasarkan warna atau tekstur, yang akan menjadi titik fokus utama dalam implementasi CBIR. Pengguna dapat dengan mudah memilih kriteria pencarian mereka dan mendapatkan hasil yang sesuai.

Ketiga, pada sisi back end menggunakan FastAPI, kami membangun API yang dapat menerima permintaan pencarian dari front end. Kami mengimplementasikan algoritma pencocokan warna dan tekstur yang efisien untuk mengidentifikasi gambar yang paling sesuai dengan kriteria yang dimasukkan oleh pengguna. Proses ini melibatkan perbandingan fitur warna dan tekstur yang diekstrak sebelumnya dari setiap gambar dalam dataset.

Selain itu, kami memastikan bahwa aplikasi ini memiliki performa yang optimal dengan memanfaatkan teknik-teknik caching untuk hasil pencarian yang sering diminta. Dengan demikian, waktu respons aplikasi dapat dipercepat, meningkatkan pengalaman pengguna.

Keempat, kami melakukan pengujian menyeluruh untuk memastikan keakuratan dan kecepatan CBIR yang diimplementasikan. Pengujian melibatkan serangkaian skenario yang mencakup variasi kriteria pencarian, ukuran dataset, dan situasi penggunaan umum. Hasil pengujian ini menjadi dasar untuk memastikan bahwa aplikasi dapat memberikan hasil pencarian yang memuaskan.

Dengan merinci langkah-langkah tersebut, diharapkan dapat memberikan pemahaman yang jelas tentang proses pengembangan aplikasi CBIR dengan fokus pada komponen warna atau tekstur menggunakan ReactJS dan FastAPI. Langkah-langkah ini juga mencakup aspek-aspek kritis seperti pra-pemrosesan data, antarmuka pengguna, implementasi algoritma

pencarian, dan pengujian aplikasi untuk memastikan kualitas dan kehandalan sistem yang dikembangkan.

B. Pemetaan Masalah pada Aljabar Geometri

Pada tahap ini, kami akan melakukan pemetaan masalah pada aljabar geometri yang terkait dengan proses Content-Based Image Retrieval (CBIR) berfokus pada komponen warna dan tekstur. Dalam implementasi CBIR, langkah-langkah kritis ini terintegrasi dengan prinsip-prinsip aljabar geometri untuk menghasilkan solusi yang sesuai dengan kebutuhan tugas besar ini.

1. Pemetaan Masalah pada CBIR Warna

Pada proses CBIR warna, kami memanfaatkan konsep aljabar geometri untuk mengonversi ruang warna RGB ke dalam model warna HSV (Hue, Saturation, Value). Proses normalisasi RGB dilakukan untuk memastikan konsistensi nilai warna antar gambar dalam dataset. Kami menghitung nilai maksimum (C_{max}) dan minimum (C_{min}) dari komponen warna RGB serta delta antara keduanya. Kemudian, kami mengonversi nilai RGB ke dalam ruang warna HSV untuk memanfaatkan tingkat kecerahan (Value) dan kejenuhan (Saturation) dalam pencarian gambar. Untuk mengukur kemiripan antara gambar, digunakan metode cosine similarity, yang juga merupakan konsep aljabar geometri.

2. Pemetaan Masalah pada CBIR Tekstur

Dalam proses CBIR tekstur, kami menggunakan prinsip-prinsip aljabar geometri untuk merinci langkah-langkah pengolahan gambar. Pertama, gambar dikonversi dari ruang warna RGB ke grayscale untuk memfokuskan analisis pada tekstur. Selanjutnya, kami membuat co-occurrence matrix yang merepresentasikan hubungan spasial antara intensitas piksel. Dengan menggunakan konsep aljabar geometri, kami mencari matriks simetris dari co-occurrence matrix dan melakukan normalisasi untuk menghasilkan distribusi probabilitas. Dari matriks tersebut, kami mengekstrak komponen-komponen seperti kontras (contrast), entropi (entropy), dan homogenitas (homogeneity), serta dissimilarity yang kemudian digunakan sebagai fitur dalam pencarian gambar.

3. Integrasi dengan Aljabar Geometri pada Front End dan Back End

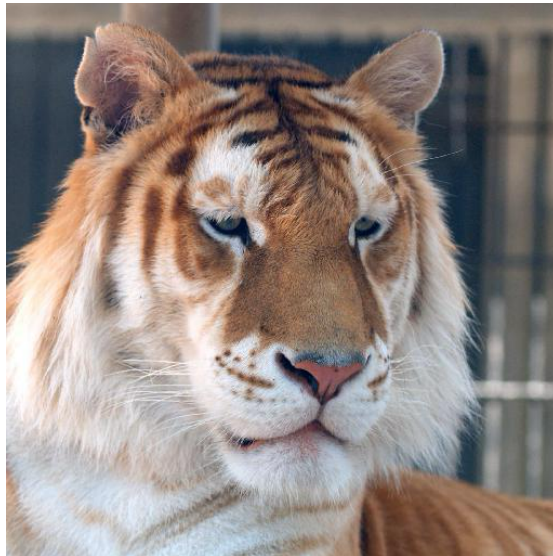
Dalam pengembangan front end menggunakan ReactJS, kami memastikan antarmuka pengguna menggambarkan informasi secara visual sesuai dengan prinsip-prinsip aljabar geometri yang terlibat dalam pemrosesan warna dan tekstur. Sebagai contoh, pemilihan kriteria pencarian warna atau tekstur pada antarmuka pengguna direpresentasikan dengan memanfaatkan elemen visual yang intuitif, seperti skala warna atau pemilihan metrik tekstur.

Pada sisi back end dengan menggunakan FastAPI, kami mengintegrasikan operasi-operasi aljabar geometri ke dalam logika pemrosesan pencarian. Implementasi algoritma cosine similarity pada proses CBIR warna dan ekstraksi fitur tekstur menggunakan co-occurrence matrix mencerminkan aplikasi prinsip-prinsip aljabar geometri yang relevan.

Melalui integrasi ini, kami meyakini bahwa penggunaan aljabar geometri dapat meningkatkan kualitas dan keakuratan sistem CBIR yang kami kembangkan, sekaligus memberikan landasan matematis yang kuat untuk setiap langkah penyelesaian masalah.

C. Ilustrasi Kasus dan Penyelesaian

Untuk memberikan gambaran, misalkan pengguna memiliki sebuah dataset gambar yang mencakup berbagai objek dan latar belakang dengan variasi warna dan tekstur yang signifikan. Seorang pengguna ingin mencari gambar yang mirip dengan gambar referensi tertentu dalam dataset ini. Contoh gambar yang dijadikan sebagai acuan adalah sebagai berikut.



Dalam proses pencarian berdasarkan warna, setiap gambar mengalami normalisasi RGB untuk memastikan konsistensi nilai warna. Selanjutnya, nilai RGB diubah ke dalam ruang warna Hue-Saturation-Value (HSV) untuk memanfaatkan kecerahan dan kejenuhan. Untuk gambar tersebut, akan didapatkan vektor HSV yaitu sebagai berikut.

```
[4.2800e+02 7.0200e+02 8.1800e+03 8.6000e+01 5.1000e+01 1.8688e+04  
3.7400e+02 1.4700e+02 5.8130e+03 2.2736e+04 2.0000e+00 1.1100e+02  
2.3492e+04 5.3150e+03 1.6660e+03 1.2500e+03 2.9150e+03 7.3000e+01  
5.1000e+01 2.1325e+04 9.3300e+02 4.7100e+02 1.1600e+04 1.6782e+04
```

1.4000e+02 1.3020e+03 2.3918e+04 3.7640e+03 2.6800e+02 3.0020e+03
 4.3010e+03 4.0400e+02 5.8100e+02 1.9628e+04 7.3000e+01 2.1000e+01
 1.9207e+04 9.0790e+03 3.0000e+00 3.1300e+02 2.8308e+04 4.0300e+02
 1.4900e+02 1.6800e+02 1.8780e+03 1.2100e+02 3.2200e+02 2.5699e+04
 2.9000e+02 8.2000e+01 1.4884e+04 1.3563e+04 3.0000e+00 0.0000e+00
 7.2320e+03 2.1668e+04 5.5900e+02 1.6060e+03 9.3600e+02 2.0400e+02
 7.6000e+01 2.4975e+04 2.5600e+02 1.2800e+02 7.6070e+03 2.0529e+04
 4.2400e+02 5.4400e+02 2.1736e+04 6.6380e+03 5.0100e+02 3.4020e+03
 6.3980e+03 2.1650e+03 2.5310e+03 1.2658e+04 1.5300e+02 5.5000e+01
 2.1062e+04 7.3710e+03 1.0900e+02 3.8000e+01 2.3445e+04 5.4260e+03
 2.2780e+03 5.6880e+03 2.9510e+03 2.8600e+02 2.4100e+02 1.5855e+04
 6.2400e+02 1.6400e+02 2.8458e+04 3.8700e+02 0.0000e+00 0.0000e+00
 1.9670e+03 2.6933e+04 1.9290e+03 5.7280e+03 5.1390e+03 5.1000e+02
 5.6500e+02 1.2693e+04 1.1890e+03 5.6000e+02 2.5204e+04 3.4090e+03
 8.3000e+01 3.1900e+02 1.5132e+04 1.3465e+04 1.4200e+02 3.2940e+03
 5.0920e+03 2.3530e+03 3.9890e+03 1.3252e+04 1.8000e+01 6.0000e+00
 1.6651e+04 1.2155e+04 0.0000e+00 0.0000e+00 2.3834e+04 5.0660e+03]

Selanjutnya, akan dilakukan hal yang sama untuk setiap gambar pada dataset. Kemiripan gambar kemudian dihitung dengan metode Cosine similarity kemudian digunakan untuk membandingkan kemiripannya dengan gambar lain.

Sementara itu, dalam pencarian berdasarkan tekstur, gambar direpresentasikan dalam skala warna grayscale untuk memfokuskan analisis pada tekstur. Matriks ko-terjadinya dibuat untuk menggambarkan hubungan spasial antara intensitas piksel, dan matriks simetris dihasilkan dari proses ini. Fitur seperti kontras, entropi, homogenitas, dan dissimilarity diekstrak dari matriks ini. Contoh hasil vektor untuk gambar di atas adalah sebagai berikut.

[202.4375, 0.22784593552207114, 2.6318529, 8.4765625]

Kemiripan gambar kemudian dihitung dengan metode Cosine similarity kemudian digunakan untuk membandingkan kemiripannya dengan gambar lain. Hasil pencarian dari kedua metode, warna dan tekstur, diberikan kepada pengguna dalam urutan dari terbesar ke terkecil dan hanya memunculkan gambar yang memiliki kemiripan diatas 60 persen. Dengan ilustrasi kasus ini, diharapkan pembaca dapat memahami secara lebih mendalam tentang bagaimana pendekatan aljabar geometri digunakan dalam CBIR untuk mencari gambar berdasarkan komponen warna dan tekstur.

BAB IV

IMPLEMENTASI DAN UJI COBA

A. Implementasi Program Utama

1. CBIR dengan Parameter Warna

a. Kuantisasi warna

```
function color_quantization(h: array, s: array, v: array, Hist:
array, iteration: integer) -> array

  KAMUS LOKAL
    h_ranges, s_ranges, v_ranges : array

  ALGORITMA
    h_ranges <- [(316, 360), (0, 25), (26, 40), (41, 120),
(121, 190), (191, 270), (271, 295), (296, 315)]
    s_ranges <- [(0, 0.2), (0.21, 0.7), (0.71, 1)]
    v_ranges <- [(0, 0.2), (0.2, 0.7), (0.7, 1)]

    i traversal [0..length(h_ranges)]
      for each (start, end) in h_ranges do
        Hist[iteration *14 + i] <- sum of h in range of
(start, end)

    i traversal [0..length(s_ranges)]
      for each (start, end) in s_ranges do
        Hist[iteration *14 + i + 8] <- sum of s in range of
(start, end)

    i traversal [0..length(v_ranges)]
      for each (start, end) in v_ranges do
        Hist[iteration *14 + i + 11] <- sum of v in range
of (start, end)

    -> Hist
```

b. Mengubah RGB menjadi HSV

```
function RGB2HSV(r: array, g: array, b: array) -> array, array,
array

  KAMUS LOKAL
    cmax, cmin, h, s, v : array
```

ALGORITMA

```
r <- r / 255  
g <- g / 255  
b <- b / 255
```

```
cmax = array of max(r,g,b)  
cmin = array of min(r,g,b)  
delta = cmax - cmin
```

```
r_indices = array of cmax == r  
g_indices = array of cmax == g  
b_indices = array of cmax == b
```

```
if h[index] = r_indices then  
  H[index] = 60 * ((g[index] - b[index]) / delta[index] % 6)  
if h[index] = g_indices then  
  H[index] = 60 * ((b[index] - r[index]) / delta[index] + 2)  
if h[index] = b_indices then  
  H[index] = 60 * ((r[index] - g[index]) / delta[index] + 4)
```

```
If cmax[index] != 0 then  
  s[index] = delta[index] / cmax[index]
```

```
-> h,s,v
```

c. Menghitung Cosinus Similarity

```
function calculate_similarity(hist1: vector, hist2: vector) -> real
```

KAMUS LOKAL

```
dot, sum1, sum2 : integer  
i :integer
```

ALGORITMA

```
i traversal [0..len(hist1)]  
  dot <- hist1[i]*hist2[i] + dot  
  
i traversal [0..len(hist1)]  
  sum1 <- hist1[i]*hist1[i] + sum1  
  
sum1 <- sqrt(sum1)  
  
i traversal [0..len(hist1)]  
  sum2 <- hist2[i]*hist2[i] + sum2  
  
sum2 <- sqrt(sum2)
```

```
-> dot / (sum1 * sum2)
```

d. Membuat array histogram dari image 3x3

```
function histogram_array (img: matrix, hists: array) -> array

KAMUS LOKAL
    height, width, start_h, end_h, start_w, end_w : integer
    R,g,b : array
    grid : matrix

ALGORITMA
    height <- height / 3
    Width <- width / 3

    i traversal (0..3)
        j traversal (0..3)
            start_h <- i * height
            end_h <- (i+1) * height
            start_w <- i * weight
            end_w <- (i+1) * weight

            grid <- image[start_h:end_h, start_w:end_w]
            r,g,b <- grid[:, :,0], grid[:, :,1] , grid[:, :,2]
            s,s,v <- RGB2HSV(r,g,b)
            hists = color_quantization(h, s, v, hists, i*3+j)

    -> hists
```

2. CBIR dengan Parameter Tekstur

a. Gambar Menjadi Grayscale

```
function imageGrayscale(RGB: image_matrix) -> matrix

KAMUS LOKAL
    i, j: integer
    gray : matrix

ALGORITMA
    i traversal [0..image_width]
        j traversal [0..image_height]
            gray[i][j] <- (RGB.R[i][j] * 0.299 +
```

```

                                RGB.G[i][j] * 0.587 + RGB.B[i][j] *
                                0.114)
-> gray

```

b. Matriks Co-Ocurrence, derajat yang digunakan adalah 0

```

function co_ocurrence(gray_matrix: matrix) -> matrix

KAMUS LOKAL
    i, j, k, l: integer
    glcm : array[0..255] of array[0..255] of integer {diisi
dengan 0}

ALGORITMA
    i traversal [0..255]
        j traversal [0..254]
            k <- gray_matrix[i][j]
            l <- gray_matrix[i][j + 1]

            glcm[k][l] <- glcm[k][l] + 1

-> glcm

```

c. Transpose Matriks

```

function transpose(M: matrix) -> matrix

KAMUS LOKAL
    i, j: integer
    N : matrix

ALGORITMA
    i traversal [0..255]
        j traversal [0..255]
            N[i][j] <- M[j][i]

-> N

```


d. Matriks Simetri

```
function simetri(M: matrix) -> matrix

KAMUS LOKAL
    i, j: integer
    N, P : matrix

ALGORITMA
    P <- transpose(M)
    i traversal [0..255]
        j traversal [0..255]
            N[i][j] <- M[i][j] + P[i][j]
    -> N
```

e. Normalisasi Matriks

```
function normalisasi(M: matrix) -> matrix

KAMUS LOKAL
    i, j, sum: integer

ALGORITMA
    sum <- 0

    i traversal [0..255]
        j traversal [0..255]
            sum <- M[i][j]

    i traversal [0..255]
        j traversal [0..255]
            M[i][j] <- M[i][j] / sum

    -> M
```

f. Ekstrak Entropi

```
function entropi(M: matrix) -> matrix

KAMUS LOKAL
    i, j, ent: integer
```

```

ALGORITMA
  ent <- 0

  i traversal [0..255]
    j traversal [0..255]
      if M[i][j] = 0 then
        continue

      ent <- M[i][j] * log(M[i][j])

  -> ent

```

g. Ekstrak Homogenitas

```

function entropi(M: matrix) -> matrix

KAMUS LOKAL
  i, j, hmg: integer

ALGORITMA
  ent <- 0

  i traversal [0..255]
    j traversal [0..255]
      hmg <- M[i][j] / (1 + (i - j) ** 2)

  -> hmg

```

h. Ekstrak Kontras

```

function entropi(M: matrix) -> matrix

KAMUS LOKAL
  i, j, con: integer

ALGORITMA
  ent <- 0

  i traversal [0..255]
    j traversal [0..255]
      con <- M[i][j] * (i - j) ** 2

  -> con

```

i. Ekstrak Dissimilarity

```
function entropi(M: matrix) -> matrix

KAMUS LOKAL
    i, j, dss: integer

ALGORITMA
    ent <- 0

    i traversal [0..255]
        j traversal [0..255]
            dss <- M[i][j] * abs(i - j)

    -> dss
```

j. Vector dari Matriks

```
function vector(img: image) -> matrix

KAMUS LOKAL
    vec : array[0..3] of integer
    gray, glcm, sim, norm : matrix
ALGORITMA
    gray <- imageGrayscale(img)
    glcm <- co_ocurrence(gray)
    sim <- simetri(glcm)
    norm <- normalisasi(sim)

    vec[0] <- kontras(norm)
    vec[1] <- homogenitas(norm)
    vec[2] <- entropi(norm)
    vec[3] <- dissimilarity(norm)

    -> vec
```

k. Cosine Similarity

```
function vector(vec1: vector, vec2: vector) -> real
```

KAMUS LOKAL

dot, sum1, sum2 : integer

i :integer

ALGORITMA

```
i traversal [0..3]
    dot <- vec1[i]*vec2[i] + dot

i traversal [0..3]
    sum1 <- vec1[i]*vec1[i] + sum1

sum1 <- sqrt(sum1)

i traversal [0..3]
    sum2 <- vec2[i]*vec2[i] + sum2

sum2 <- sqrt(sum2)

-> dot / (sum1 * sum2)
```

B. Struktur Program

Struktur program dibuat menjadi 2 yaitu back-end dan front-end. Tujuan dari dipisahkannya kedua folder tersebut adalah memudahkan pembagian tugas dan agar lebih mudah dipahami.

1. Front End

```
|-- app/
|   |-- about/
|   |   |-- layout.tsx
|   |   |-- page.tsx
|   |
|   |-- camera/
|   |   |-- layout.tsx
|   |   |-- page.tsx
|   |
|   |-- cbir/
|   |   |-- layout.tsx
|   |   |-- page.tsx
|   |
|   |-- layout.tsx
|   |-- page.tsx
|
```

```

|-- assets/
|   |-- background/
|   |   |-- background1.jpg
|   |   |-- background2.jpg
|   |
|   |-- image/
|   |   |-- image1.png
|   |   |-- image2.png
|   |
|-- components/
|   |-- commonComponent1.tsx
|   |-- commonComponent2.tsx
|   |-- ...

```

Dalam direktori "app," terdapat beberapa sub-direktori, seperti "about," "camera," dan "cbir." Setiap sub-direktori terdapat dua file, "layout.tsx" dan "page.tsx,". Layout berfungsi untuk mengatur kemunculan komponen yang ada di page, sedangkan file page merupakan isi page itu sendiri. Selain sub-direktori page, terdapat juga dua file, "layout.tsx" dan "page.tsx," yang berada langsung di dalam direktori "app.tsx" yang berfungsi sebagai home page web.

Direktori "assets" berisi dua sub-direktori, yaitu "background" dan "image." Sub-direktori "background" berisi gambar latar, sedangkan sub-direktori "image" berisi gambar lainnya, seperti foto profil ataupun elemen visual lain dalam web. Terakhir, direktori "components" berisi beberapa file. File-file tersebut berisi komponen-komponen umum yang dapat digunakan kembali di berbagai bagian web.

2. Backend

```

|-- uploads/
|   |-- data-set/
|   |   |-- image1.jpg
|   |   |-- image2.jpg
|   |   |-- . . . .
|   |
|   |-- search/
|   |   |-- received_image.jpg
|   |
|
|-- caches.py
|-- ExportPDF.py
|-- CBIR.py

```

```
|-- CBIR_multiprocess.py  
|-- texture.py  
|-- main.py
```

Folder back-end berisi folder “uploads” yang didalamnya berisi folder “data-set” dan “search”. Folder dataset berfungsi untuk menampung gambar-gambar dataset yang diupload oleh user, sedangkan folder search akan meampung gambar yang dijadikan acuan pencarian.

Selain itu, terdapat file cache yang berfungsi untuk membuat cache, file exportPDF yang berfungsi untuk mengubah result menjadi pdf. Fungsi CBIR dengan parameter color disimpan dalam file CBIR sedangkan untuk parameter texture disimpan dalam file texture. Terakhir ada file “main.py” yang berisi seluruh konfigurasi back-end, mulai dari pengaturan end-point hingga fungsi untuk memproses request dan front-end.

C. Penggunaan Program




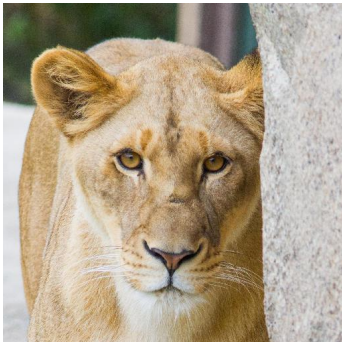
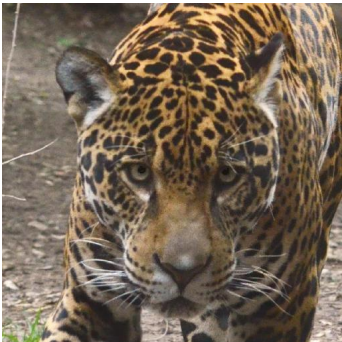
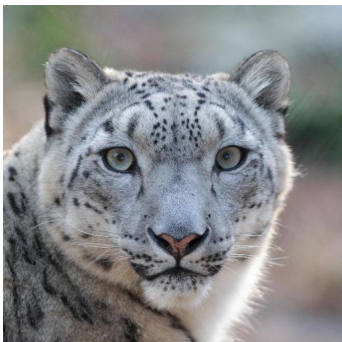
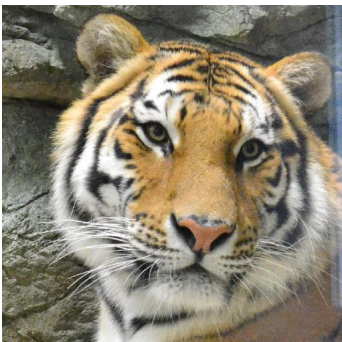
1. Mencari Gambar Menggunakan Gambar Lain.

- l. Masukkan gambar permintaan dengan menekan tombol unggah di bagian Input Gambar.
- m. Gambar ini akan menjadi gambar utama yang akan dibandingkan dengan dataset.
- n. Tekan tombol unggah dan pilih folder yang berisi dataset yang telah disiapkan atau user bisa menggunakan fitur image scraping dengan hanya melakukan submit sebuah link.
- o. Pilih antara metode pencarian berdasarkan warna atau tekstur dan tunggu hingga hasil muncul pada bagian result.

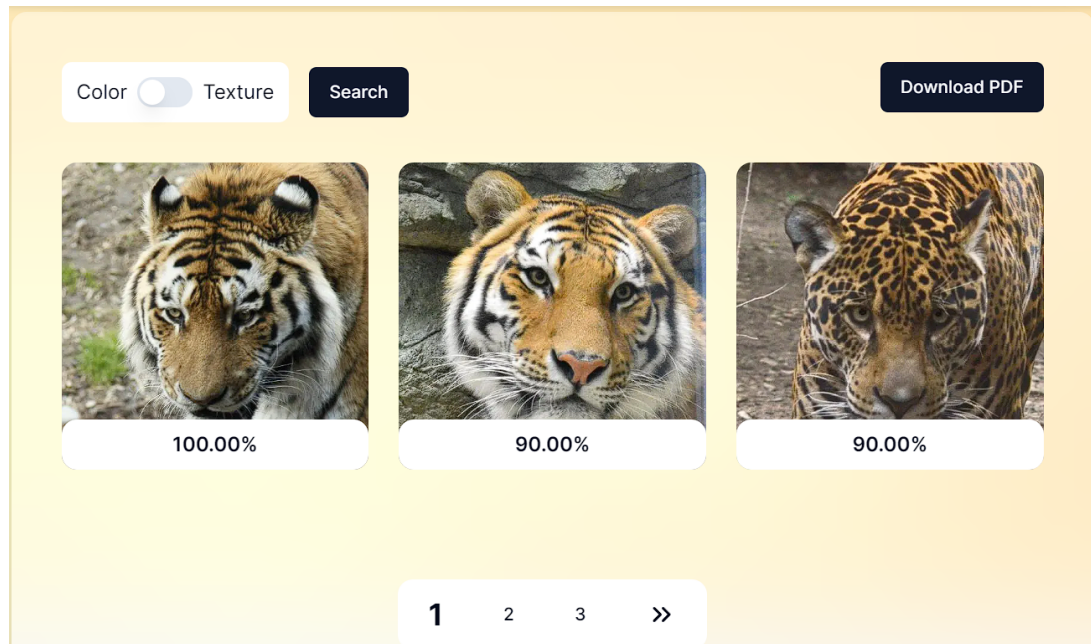
2. Mencari Gambar Menggunakan Kamera

- a. Masuk ke bagian *Camera* pada navigation bar dan allow access untuk kamera apabila ada notifikasi.
- b. Gambar akan diambil setiap 5 detik dan akan dijadikan gambar utama yang akan dibandingkan dengan dataset.
- c. Tekan tombol unggah dan pilih folder yang berisi dataset yang telah disiapkan atau user bisa menggunakan fitur image scraping dengan hanya melakukan submit sebuah link.
- d. Pilih antara metode pencarian berdasarkan warna atau tekstur dan hasil akan muncul setiap 5 detik sesuai dengan hasil foto kamera yang diupload.

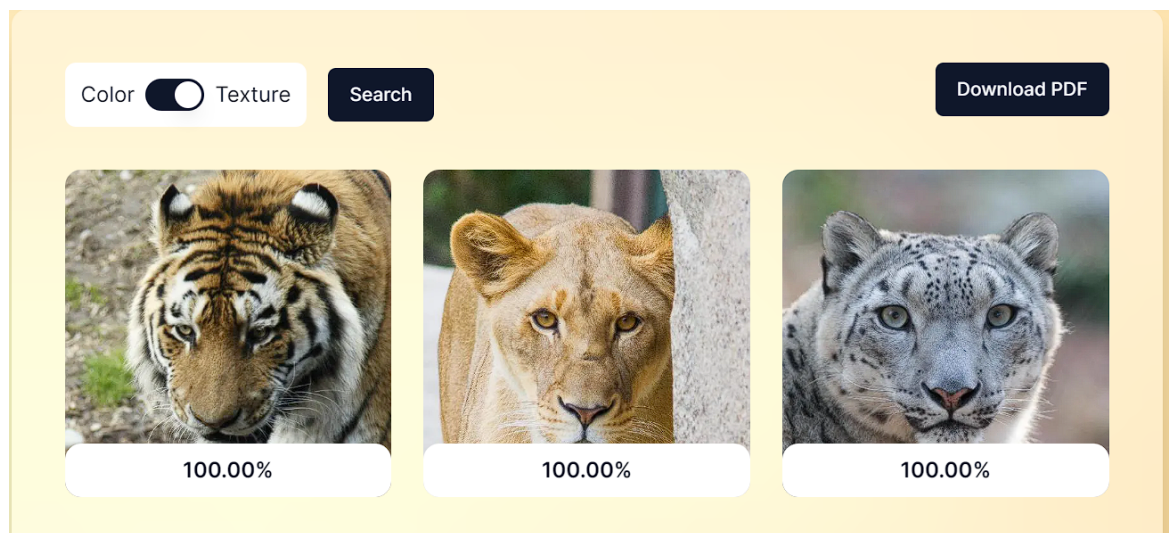
D. Hasil Pengujian

Query	Dataset	
		
		
		

1. Uji coba 1 (Color)



2. Uji coba 2 (Texture)



E. Analisis Desain Solusi

Pada implementasi Content-Based Image Retrieval (CBIR) dengan parameter color, hasil yang diperoleh sangat memuaskan. Sistem mampu memproses sebanyak 4000 gambar dalam waktu singkat, hanya dalam 60 detik. Kecepatan ini memberikan pengguna pengalaman yang baik dalam mencari gambar berdasarkan warna. Lebih lanjut, persentase kemiripan antara gambar yang dihasilkan dengan gambar referensi juga mencapai tingkat

akurasi yang tinggi, menunjukkan kehandalan algoritma color-based retrieval yang diimplementasikan.

Sementara itu, pada parameter tekstur, kecepatan pemrosesan tetap konsisten dengan parameter warna. Namun, terdapat perbedaan signifikan pada hasilnya. Meskipun kecepatan tetap tinggi, tingkat kemiripan gambar berdasarkan tekstur sering kali memberikan hasil yang sangat tinggi, terutama jika dibandingkan dengan gambar yang memiliki pola tekstur yang sederhana atau polos. Hal ini menunjukkan keefektifan algoritma tekstur dalam mengekstraksi ciri-ciri yang kompleks dari gambar, memberikan hasil yang sangat relevan dengan kriteria pencarian pengguna.

Namun, tantangan yang masih dihadapi adalah kurangnya validasi terhadap tipe file yang diunggah oleh pengguna. Belum adanya penanganan khusus untuk file selain gambar dapat menjadi potensi sumber masalah dan perlu ditangani dengan baik untuk meningkatkan pengalaman pengguna. Selain itu, fitur-fitur tambahan seperti ekspor ke format PDF dan image scraping telah diimplementasikan dengan sukses, memberikan nilai tambah yang signifikan pada sistem ini.

Secara keseluruhan, sistem CBIR yang diimplementasikan menggunakan ReactJs sebagai frontend dan FastAPI sebagai backend telah memberikan hasil yang positif. Kecepatan pemrosesan yang tinggi dan hasil pencarian yang relevan menunjukkan keberhasilan implementasi algoritma. Dengan menangani beberapa kekurangan, seperti validasi tipe file dan penanganan kasus khusus, sistem ini dapat menjadi solusi yang komprehensif dan handal untuk kebutuhan Content-Based Image Retrieval pada platform web.

BAB V

KESIMPULAN

A. Kesimpulan

Dari hasil pengujian kedua algoritma CBIR, dapat disimpulkan bahwa keduanya bekerja dengan sangat cepat dan efisien. Algoritma CBIR dengan parameter warna menghasilkan hasil sesuai dengan harapan, sementara algoritma dengan parameter tekstur cenderung menghasilkan angka kemiripan yang tinggi, seringkali mencapai 99%. Selain itu, integrasi antara React dan FastApi juga menunjukkan kinerja yang baik, menunjukkan bahwa keduanya dapat berinteraksi dengan efisien satu sama lain.

B. Saran

Meskipun algoritma CBIR dengan parameter warna memberikan hasil yang memuaskan, disarankan untuk mencoba menggunakan algoritma lain dalam proses perhitungan kemiripan untuk parameter tekstur. Dengan mencoba variasi algoritma, mungkin dapat ditemukan pendekatan yang lebih baik untuk menangani kasus-kasus di mana angka kemiripan cenderung selalu tinggi.

Selain itu, perlu diperhatikan untuk menangani beberapa bug yang mungkin muncul selama pengembangan. Penanganan bug yang lebih baik akan meningkatkan stabilitas dan kehandalan sistem secara keseluruhan. Dalam hal pengembangan web, beberapa komponen seharusnya dapat dibagi menjadi unit yang lebih kecil untuk memudahkan organisasi dan pemeliharaan sistem. Ini dapat membantu tim pengembangan dalam memahami dan mengelola kode dengan lebih efisien.

C. Komentar

Secara umum, tugas besarnya keren *sih*. Tugasnya ememperluas pengetahuan tentang pengembangan web, terutama tentang komunikasi antara backend dan frontend.

D. Refleksi

Dalam refleksi terhadap tugas besar ini, perencanaan awal yang matang terbukti penting. Dengan merencanakan dengan baik UI dan fitur-fitur yang akan dimasukkan sejak awal, dapat membantu menghindari perubahan besar yang mungkin diperlukan di tengah jalan. Selama pengerjaan, penambahan fitur sering kali memerlukan penggantian kode, yang bisa menjadi tantangan tersendiri.

Pemahaman ini dapat menjadi pelajaran berharga untuk proyek-proyek mendatang, menekankan pentingnya perencanaan dan desain yang matang sejak awal untuk mencapai efisiensi dan kualitas yang lebih baik dalam pengembangan perangkat lunak.

DAFTAR PUSTAKA

- Ramadhani, M. F. (2015). Pembangunan Aplikasi Informasi, Pengaduan, Kritik, Dan Saran Seputar Kota Cimahi Pada Platform Android. Jurnal Ilmiah Komputer Dan Informatika (KOMPUTA), 9.
- Pranata, B. A., Hijriani, A., & Junaidi, A. (2018). Perancangan Application Programming Interface (API) Berbasis WEB Menggunakan Gaya Arsitektur Representational State Transfer (REST) Untuk Pengembangan Sistem Informasi Administrasi Pasien Klinik Perawatan Kulit. Jurnal Komputasi, 6(1), 33-42.
- Kornienko, D. V., Mishina, S. V., & Melnikov, M. O. (2021). Principles of securing RESTful API web services developed with python frameworks. Journal of Physics: Conference Series, 1-11.
- Dewi, M., Fauriatun, H., & Nurul, R. (2021). Penyuluhan Manfaat Menggunakan Internet dan Website Pada Masa Pandemi Covid-19. Jurnal Pengabdian Masyarakat Informatika, 1-7.

Github repository: <https://github.com/maulvi-zm/Algeo02-22122>