

LAPORAN TUGAS BESAR II
IF2123 ALJABAR LINEAR DAN GEOMETRI
Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar



Disusun oleh:

Enrique Yanuar (13522077)
Naufal Adnan (13522116)
Mesach Harmasendro (13522117)

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

2022

DAFTAR ISI

DAFTAR ISI.....	1
BAB I.....	2
BAB II.....	3
I. Representasi citra digital sebagai matriks.....	3
II. <i>Content-Based Image Retrieval (CBIR)</i>	3
III. <i>Co-occurrence matrix</i>	4
IV. CBIR dengan parameter warna.....	5
V. CBIR dengan parameter tekstur.....	7
VI. Website CBIR (VisuMatch).....	8
BAB III.....	10
I. Langkah-langkah pemecahan masalah.....	10
II. Pemetaan masalah-masalah ke elemen aljabar geometri.....	15
III. Ilustrasi kasus dan contoh penyelesaiannya.....	16
BAB IV.....	20
I. Implementasi.....	20
II. Struktur program.....	30
III. Penggunaan program.....	33
IV. Hasil pengujian.....	47
V. Analisis desain solusi algoritma CBIR.....	67
BAB V.....	68
I. Kesimpulan.....	68
II. Saran.....	68
III. Komentar atau tanggapan.....	68
IV. Refleksi.....	69
V. Ruang perbaikan atau pengembangan.....	69
DAFTAR PUSTAKA.....	71
LAMPIRAN.....	72

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 adalah Google Lens.

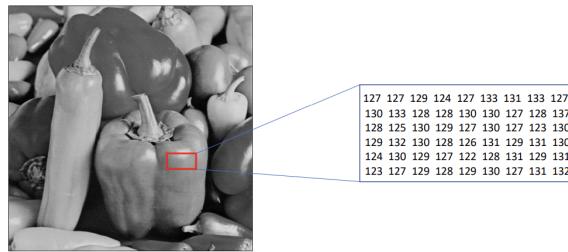
Dari permasalahan tersebut akan dibuat implementasi sistem temu balik gambar dengan memanfaatkan aljabar vektor dalam bentuk sebuah *website*, dimana hal ini merupakan pendekatan yang penting dalam dunia pemrosesan data dan pencarian informasi. Dalam konteks ini, aljabar vektor digunakan untuk menggambarkan 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

I. Representasi citra digital sebagai matriks

Gambar yang kita lihat di layar komputer direpresentasikan sebagai matriks berukuran $M \times N$, dimana $M \times N$ menyatakan resolusi dari citra atau gambar. Setiap elemen matriks tersebut menyatakan sebuah pixel (*picture element*). Suatu gambar hitam putih tersusun atas elemen dari matriks $M \times N$ berupa angka-angka integer [0..255]. Angka-angka tersebut merepresentasikan warna dari hitam (representasi angka 0) hingga putih (representasi angka 255). Berikut ilustrasi representasi citra digital sebagai matriks *grayscale*.



Pada gambar berwarna, citra dapat direpresentasikan dengan model matriks RGB. Model ini menggunakan 3 buah matriks gambar hitam-putih yang masing-masing merepresentasikan warna merah (Red), hijau (Green), biru (Blue). Pada matriks R (red), warna merah sempurna direpresentasikan dengan nilai 255 dan hitam sempurna dengan nilai 0, begitu juga pada matriks G (green), dan B (blue). Ketiga matriks tersebut jika disatukan akan menghasilkan gambar berwarna sehingga setiap pixel pada gambar RGB ini memiliki intensitas warna yang merupakan kombinasi dari tiga nilai intensitas pada matriks R, G, dan B.

Pemrosesan gambar dapat dilakukan dengan memanfaatkan berbagai properti yang dimiliki oleh matriks seperti melakukan transformasi pada gambar, *image scaling*, *bicubic interpolation*, *image retrieval*, dan penerapan lainnya.

II. Content-Based Image Retrieval (CBIR)

Content-Based Image Retrieval (CBIR) merupakan sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan konten visual gambar seperti warna, tekstur, dan bentuk, tanpa bergantung pada kata kunci atau teks. Hal ini

membuat CBIR sangat berguna dalam mengakses dan mengeksplorasi koleksi gambar ketika tidak ada informasi teks atau kata kunci yang cukup untuk mendeskripsikan gambar yang dicari. Fitur-fitur penting dari gambar, seperti warna, tekstur, dan bentuk akan diekstraksi terlebih dahulu sebelum selanjutnya akan diwakili dalam bentuk vektor atau deskripsi numerik. Lalu CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan selanjutnya dapat digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling sesuai dengan gambar yang dicari. Implementasi CBIR yang paling populer digunakan adalah CBIR dengan parameter warna dan CBIR dengan parameter tekstur.

III. Co-occurrence matrix

Co-occurrence matrix atau *co-occurrence distribution* merupakan sebuah matriks yang didefinisikan pada suatu gambar sebagai distribusi nilai-nilai piksel yang terjadi bersamaan pada *offset* tertentu. Matriks *co-occurrence* dari suatu gambar mewakili jarak dan hubungan spasial sudut pada subwilayah gambar dengan ukuran tertentu. 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 berikut.

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

Dengan mempertimbangkan intensitas atau nilai skala abu-abu gambar atau berbagai dimensi warna, matriks *co-occurrence* dapat mengukur tekstur gambar. Melalui persamaan tersebut, digunakan nilai θ adalah $0^\circ, 45^\circ, 90^\circ$, dan 135° . Berikut ilustrasi pembuatan *co-occurrence matrix*.

	1	2	3	4	5	6	7	8
1	1	2	0	0	1	0	0	0
2	0	0	1	0	1	0	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0	0
5	1	0	0	0	0	1	2	0
6	0	0	0	0	0	0	0	1
7	2	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0

IV. CBIR dengan parameter warna

CBIR dengan parameter warna dilakukan dengan cara mengubah gambar yang berbentuk RGB menjadi sebuah metode histogram warna yang lebih umum, yang kemudian akan dibandingkan masukan dari sebuah gambar dengan gambar yang dimiliki oleh dataset. Histogram warna merupakan frekuensi dari berbagai warna dalam suatu ruang warna tertentu, bertujuan untuk menggambarkan pendistribusian warna dari suatu gambar. Histogram warna tidak bisa mengidentifikasi objek spesifik yang terdapat pada gambar dan tidak bisa memberikan deskripsi tentang lokasi dari warna yang didistribusikan.

Untuk membuat sebuah histogram warna, dimana setiap interval tiap *range*-nya dianggap sebagai *bin*, dilakukan pembentukan ruang warna. Pembentukan ruang warna diperlukan dalam rangka melakukan pembagian nilai citra menjadi beberapa *range* yang lebih kecil. Histogram warna dapat dihitung dengan menghitung piksel yang mewakili nilai warna pada setiap interval. Fitur warna mencakup histogram warna global dan histogram warna blok. Pada perhitungan histogram, penggunaan warna global HSV lebih dipilih karena warna tersebut dapat digunakan pada kertas yang lebih umum untuk digunakan, yaitu kertas dengan *background* putih.

Berdasarkan hal tersebut, maka diperlukan konversi warna dari RGB ke HSV dengan langkah-langkah sebagai berikut.

1. Normalisasikan nilai dari RGB, yaitu dengan mengubah nilai *range* dari [0, 255] menjadi [0, 1]

$$R' = \frac{R}{255}$$

$$G' = \frac{G}{255}$$

$$B' = \frac{B}{255}$$

2. Cari nilai *Cmax*, *Cmin*, dan Δ

$$Cmax = \max(R', G', B')$$

$$Cmin = \min(R', G', B')$$

$$\Delta = Cmax - Cmin$$

3. Hitung nilai HSV

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

$$S = \left\{ \begin{array}{ll} 0 & , Cmax = 0 \\ \frac{\Delta}{Cmax} & , Cmax \neq 0 \end{array} \right\}$$

$$V = Cmax$$

Dari nilai HSV yang telah diperoleh, selanjutnya dilakukan perbandingan antara gambar dari input dengan dataset menggunakan *cosine similarity*

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

n : jumlah dimensi dari vektor

A dan B : vektor dari kedua gambar

Besar hasil dari *cosine similarity* menunjukkan tingkat kemiripan kedua gambar. Semakin besar nilai *cosine similarity*, maka tingkat kemiripannya semakin tinggi. Pencarian histogram dilakukan dengan membagi block image menjadi $n \times n$ blok. Jika blok-blok tersebut terlalu besar, maka setiap blok akan menjadi tidak terlalu signifikan. Namun, jika ukuran dari blok tersebut terlalu kecil, maka akan meningkatkan waktu dalam memprosesnya. Agar lebih efektif, maka lebih dipilih blok dengan 4×4 blok. Sebuah blok akan dinyatakan nilai representatifnya dengan melakukan kalkulasi nilai rata-rata HSV dari blok terkait.

V. CBIR dengan parameter tekstur

Berbeda dari CBIR dengan parameter warna. CBIR dengan parameter tekstur tidak melibatkan representasi warna. CBIR dengan parameter tekstur dilakukan dengan memanfaatkan *co-occurrence matrix*. Setelah *co-occurrence matrix* diperoleh, dibuatlah *symmetric matrix* dengan cara menjumlahkan *co-occurrence matrix* dengan hasil transposenya. Kemudian mencari nilai *matrix normalization* dengan persamaan berikut.

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

Prosedur dalam CBIR dengan parameter tekstur adalah sebagai berikut.

1. Warna tidaklah penting dalam penentuan tekstur sehingga diperlukan konversi warna gambar menjadi *grayscale*. Warna RGB dapat diubah menjadi suatu warna *grayscale* Y dengan rumus sebagai berikut.

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

2. Lakukan kuantifikasi dari nilai *grayscale*. Matriks yang berkoresponden akan berukuran 256×256 karena citra *grayscale* berukuran 256 piksel. Berdasarkan penglihatan manusia, tingkat kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.
3. Dari *co-occurrence matrix* dapat diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity* dengan persamaan sebagai berikut.

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 :

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

dengan P merupakan matriks *co-occurrence*.

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

4. Ukur kemiripan dari kedua gambar menggunakan *cosine similarity*, yaitu:

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

n : jumlah dimensi dari vektor

A dan B : vektor dari kedua gambar

Besar hasil dari *cosine similarity* menunjukkan tingkat kemiripan kedua gambar. Semakin besar nilai *cosine similarity*, maka tingkat kemiripannya semakin tinggi.

VI. Website CBIR (VisuMatch)

Pengembangan sebuah *website* melibatkan proses mulai dari merancang, membangun, dan mengoptimasi situs web agar dapat berfungsi secara efisien dan memberikan pengalaman pengguna yang baik. Perencanaan meliputi kegiatan identifikasi audiens target, pemilihan platform, dan penentuan fitur serta fungsionalitas yang dibutuhkan. Desain website mencakup pembuatan tata letak, grafik, warna, dan elemen visual lainnya. Setelah itu, pengembangan *front-end* akan dilakukan untuk membangun elemen-elemen yang dapat dilihat dan diakses oleh pengguna. *Front-end* ini melibatkan pengkodean dan implementasi desain ke dalam HTML, CSS, dan JavaScript. Lalu pengembangan *back-end* melibatkan pembuatan dan pengelolaan server, database, dan logika bisnis. Sisi *back-end* membuat fitur-fitur yang tidak terlihat oleh pengguna, seperti pemrosesan formulir, manajemen database, dan fungsionalitas server-side. Kemudian integrasi database diperlukan jika situs web melibatkan penggunaan database. Di sini pengembang *back-end* akan merancang dan mengintegrasikan struktur database untuk menyimpan dan mengelola data. Sebelum situs web diluncurkan, uji dan debugging dilakukan untuk memastikan bahwa semua fungsi berjalan dengan baik dan tidak ada kesalahan atau bug yang dapat memengaruhi pengalaman pengguna. Setelah situs web diuji dan dianggap siap, *website* dapat diluncurkan dan diakses oleh pengguna. Situs web memerlukan

pemeliharaan terus-menerus. Pemeliharaan ini mencakup pembaruan keamanan, perbaikan bug, peningkatan fitur, dan penyesuaian lainnya sesuai kebutuhan.

Pengembangan sebuah *website* Content-Based Image Retrieval (CBIR) melibatkan langkah-langkah perencanaan, desain, pengembangan front-end dan back-end, perancangan algoritma CBIR, lalu pengujian. *Website* dibangun dengan antarmuka pengguna yang memungkinkan pengguna untuk mengunggah dataset gambar lalu memasukkan gambar pencarian dan menampilkan hasil berdasarkan kesamaan konten visual dengan parameter warna atau tekstur. Ekstraksi fitur gambar akan dijalankan ketika dataset diunggah ke dalam web. Informasi penting dari gambar seperti warna, tekstur, dan bentuk akan disimpan dalam representasi matriks dari fitur-fitur yang telah diekstraksi untuk setiap gambar dalam basis data. Representasi gambar kemudian akan disimpan dalam basis data untuk efisiensi pencarian. Untuk melakukan pencarian gambar berdasarkan konten digunakan algoritma pencocokan untuk membandingkan fitur gambar pencarian dengan fitur gambar dalam basis data. Hasil dari pencocokan selanjutnya digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling sesuai dengan gambar yang dicari. Optimasi kinerja pada ekstraksi fitur-fitur dan algoritma pencocokan dilakukan untuk memastikan pencarian gambar berlangsung cepat dan efisien. Penting untuk memastikan bahwa *website* CBIR dapat memberikan hasil pencarian yang akurat dan efisien berdasarkan konten visual gambar, memfasilitasi pengguna dalam mengeksplorasi, dan mengakses koleksi gambar dengan lebih efektif. Untuk meningkatkan pengalaman pengguna yang lebih baik, disediakan antarmuka pengguna yang responsif dan ramah pengguna.

BAB III

ANALISIS PEMECAHAN MASALAH

I. Langkah-langkah pemecahan masalah

Dari deskripsi persoalan masalah yang telah diuraikan pada BAB I, langkah-langkah pemecahan masalah dapat didekomposisikan sebagai berikut.

1. Perencanaan dan perancangan

Kebutuhan pengguna perlu diidentifikasi terlebih dahulu untuk menentukan tujuan yang akan dicapai dan batasannya sehingga perencanaan dan perencanaan akan lebih terstruktur. Pada persoalan ini, akan dibuat implementasi sistem temu balik gambar dengan memanfaatkan aljabar vektor dalam bentuk sebuah *website*. Pencarian akan dilakukan berdasarkan parameter warna atau tekstur.

2. Ekstraksi fitur

Dalam pemrosesannya, sistem temu balik gambar pada persoalan ini memiliki parameter pencarian berdasarkan warna atau tekstur. Persoalan ini mengharuskan informasi dari dataset yang telah diunggah perlu diekstraksi terlebih dulu, terkait informasi warna dan teksturnya. Hasil ekstraksi gambar ini akan direpresentasikan dalam bentuk matriks sebelum selanjutnya akan dilakukan pemrosesan gambar dengan memanfaatkan berbagai properti yang dimiliki oleh matriks.

a. Ekstraksi fitur warna

Untuk parameter pencarian berdasarkan warna akan diambil informasi mengenai warna pada gambar dengan cara mengubah gambar yang berbentuk RGB menjadi sebuah metode histogram warna yang lebih umum. Histogram yang dipilih adalah HSV karena warna tersebut dapat digunakan pada kertas yang lebih umum untuk digunakan, yaitu kertas dengan *background* putih. Implementasi konversi warna dari RGB ke HSV dilakukan dengan langkah-langkah sebagai berikut.

1. Normalisasikan nilai dari RGB, yaitu dengan mengubah nilai *range* dari [0, 255] menjadi [0, 1]

$$R' = \frac{R}{255}$$

$$G' = \frac{G}{255}$$

$$B' = \frac{B}{255}$$

2. Cari nilai $Cmax$, $Cmin$, dan, Δ

$$Cmax = \max (R', G', B')$$

$$Cmin = \min (R', G', B')$$

$$\Delta = Cmax - Cmin$$

3. Hitung nilai HSV

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

$$S = \left\{ \begin{array}{ll} 0 & , Cmax = 0 \\ \frac{\Delta}{Cmax} & , Cmax \neq 0 \end{array} \right\}$$

$$V = Cmax$$

b. Ekstraksi fitur tekstur

Sementara untuk parameter pencarian berdasarkan tekstur akan diambil informasi mengenai *contrast*, *entropy* dan *homogeneity* dengan mengimplementasikan *co-occurrence* matriks. Ekstraksi fitur tekstur ini dapat dilakukan dengan langkah-langkah berikut.

1. Warna tidaklah penting dalam penentuan tekstur sehingga diperlukan konversi warna gambar menjadi *grayscale*. Warna RGB dapat diubah menjadi suatu warna *grayscale* Y dengan rumus sebagai berikut.

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

2. Kuantifikasi nilai *grayscale*. Matriks yang berkoresponden akan berukuran 256×256 karena citra *grayscale* berukuran 256 piksel. Berdasarkan penglihatan manusia, tingkat

kemiripan dari gambar dinilai berdasarkan kekasaran tekstur dari gambar tersebut.

3. Dari *co-occurrence matrix* dapat diperoleh 3 komponen ekstraksi tekstur, yaitu *contrast*, *entropy* dan *homogeneity* dengan persamaan sebagai berikut.

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 :

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

dengan P merupakan matriks *co-occurrence*.

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

3. Algoritma pencocokan

Setelah diperoleh vektor fitur, baik dengan parameter warna maupun tekstur dalam dataset, pengguna akan memasukkan query berupa gambar. Query juga akan diekstraksi sehingga diperoleh vektor-fiturnya. Selanjutnya algoritma pencocokan digunakan untuk membandingkan vektor-fitur dari query dengan vektor-fitur gambar dalam dataset. Pengukuran kemiripan dari kedua gambar dilakukan dengan menggunakan *cosine similarity*, yaitu:

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

n : jumlah dimensi dari vektor

A dan B : vektor dari kedua gambar

Besar hasil dari *cosine similarity* menunjukkan tingkat kemiripan kedua gambar. Semakin besar nilai *cosine similarity*, maka tingkat kemiripannya semakin tinggi.

4. *Front-End*

Untuk mengembangkan sebuah website diperlukan antar-muka yang responsif dan ramah pengguna untuk meningkatkan pengalaman pengguna yang lebih baik. *Front-end* menyediakan cara bagi pengguna untuk berinteraksi dengan situs web mencakup segala sesuatu mulai dari mengklik tombol, hingga menggeser atau menyentuh tombol atau fitur lainnya. *Website* dibangun dengan antarmuka pengguna yang memungkinkan pengguna untuk mengunggah dataset gambar lalu memasukkan gambar pencarian dan menampilkan hasil berdasarkan kesamaan konten visual dengan parameter warna atau tekstur. Persoalan ini juga berkaitan dengan *pagination* dari pencarian gambar yang dihasilkan. Fitur lain juga pengguna dapat melakukan pengunduhan dari pencarian kesamaan gambar yang dihasilkan dengan format pdf.

5. *Back-End*

Sisi *back-end* akan berperan dalam pemrosesan yang menentukan cara situs web ini berfungsi, algoritma pencocokan pada CBIR, memanajemen database, serta koneksi dengan *front-end*. Manajemen database meliputi penyimpanan informasi penting dari gambar seperti warna, tekstur, dan bentuk dalam representasi matriks dari fitur-fitur yang telah diekstraksi untuk setiap gambar dalam dataset yang diunggah. *Back-end* terkoneksi dengan *front-end* dengan memberikan API (Application Programming Interface) untuk memungkinkan frontend berkomunikasi dengan *back-end*. *Back-end* akan menangani permintaan pengguna dengan merespons permintaan dari frontend.

6. *Caching Result*

Caching result dari perhitungan sebuah dataset dilakukan dengan menggunakan *output file* tertentu yaitu json. Hal ini akan meningkatkan efisiensi dan efektifitas ketika suatu dataset hendak digunakan berulang kali. Proses ini membantu menghindari redundansi perhitungan yang mahal secara komputasi dan memungkinkan penggunaan kembali hasil perhitungan yang telah disimpan, menghemat waktu dan sumber daya komputasi. Daripada mengulangi proses ini setiap kali dataset digunakan, caching memungkinkan

untuk menyimpan hasil perhitungan dalam file json, yang dapat dengan cepat diakses kembali saat diperlukan. Sebelum memulai perhitungan, sistem akan memeriksa apakah hasil perhitungan untuk dataset tertentu sudah ada dalam file json. Jika sudah ada, langkah perhitungan dapat dilewati dan hasilnya dapat langsung diambil dari cache. Jika hasil perhitungan belum tersedia, sistem akan menjalankan perhitungan sesuai kebutuhan. Setelah selesai, hasil perhitungan tersebut akan disimpan dalam format json yang sesuai dengan struktur data. Saat dataset yang sama diperlukan kembali, sistem akan lebih dulu mencari hasil perhitungan dalam file json. Jika ditemukan, hasil tersebut dapat langsung digunakan, menghindari perhitungan ulang. Untuk memastikan bahwa cache tetap relevan, diperlukan mekanisme manajemen cache. Hal ini dapat mencakup pembersihan berkala dari cache yang tidak digunakan.

7. Fitur Kamera

Fitur ini meningkatkan pengalaman pengguna dengan memungkinkan aplikasi untuk terus-menerus memantau dan merekam gambar dari webcam saat program berjalan. Pengguna dapat melihat hasil tangkapan gambar langsung. Program akan dimulai dengan menginisialisasi kamera pada saat peluncuran. Setelah kamera diinisialisasi, program akan mengatur interval waktu di mana penangkapan gambar dilakukan secara otomatis, yaitu setelah 10 detik, program akan mengambil gambar dari webcam. Hasil tangkapan gambar akan ditampilkan secara real-time pada antarmuka program. Kemudian akan dicari kemiripan gambar hasil tangkapan dengan dataset yang ada. Meskipun fitur kamera beroperasi secara terpisah, implementasinya harus disinkronkan dengan fitur utama program, seperti penggunaan gambar yang diunggah melalui website. Informasi atau data yang diperoleh dari tangkapan gambar webcam diintegrasikan dengan fitur utama untuk memberikan pengguna informasi yang lebih lengkap.

8. *Image Scraping*

Proses *image scraping* dilakukan dengan menggunakan Chrome DevTools Protocol (Chrome DP) untuk mengekstraksi gambar dari sebuah situs web tertentu. Hal ini memungkinkan program mengambil gambar secara otomatis tanpa interaksi pengguna tambahan. Program akan dimulai dengan menginisialisasi Chrome DP, yang memungkinkan komunikasi langsung

dengan browser Google Chrome. Proses ini melibatkan pembukaan browser Chrome dalam mode tersembunyi atau tanpa antarmuka pengguna. Setelah Chrome DP diinisialisasi, program akan menggunakan protokol tersebut untuk membuka dan menavigasi ke situs web yang menjadi target untuk scraping. Hal ini memungkinkan program untuk mengakses dan mengeksplorasi halaman web secara otomatis. Program akan menggunakan Chrome DP untuk mengekstraksi gambar dari halaman web target. Proses ini melibatkan identifikasi elemen HTML yang berisi gambar, mengunduhnya, dan menyimpannya dalam direktori atau struktur data yang sesuai. Hasil scraping, yaitu gambar-gambar yang berhasil diekstraksi, akan diintegrasikan ke dalam dataset gambar yang digunakan oleh program. Proses ini melibatkan penyimpanan lokasi file gambar dan informasi terkait lainnya ke dalam struktur dataset yang ada.

II. Pemetaan masalah-masalah ke elemen aljabar geometri

1. Pemetaan citra dalam matriks RGB sebagai representasi warna

Terdapat relasi antara matriks dengan gambar. Gambar yang ada pada layar komputer direpresentasikan sebagai matriks berukuran $M \times N$, dimana $M \times N$ menyatakan resolusi dari citra atau gambar. Setiap elemen matriks tersebut menyatakan sebuah pixel (*picture element*). Pada ekstraksi fitur warna digunakan 3 buah matriks *grayscale* yang masing-masing merepresentasikan warna merah (Red), hijau (Green), biru (Blue). Pada matriks R (red), warna merah sempurna direpresentasikan dengan nilai 255 dan hitam sempurna dengan nilai 0, begitu juga pada matriks G (green), dan B (blue). Ketiga matriks tersebut jika disatukan akan menghasilkan gambar berwarna sehingga setiap pixel pada gambar RGB ini memiliki intensitas warna yang merupakan kombinasi dari tiga nilai intensitas pada matriks R, G, dan B.

2. Konversi parameter warna dari RGB ke histogram yang lebih umum.

Setelah gambar direpresentasikan dalam matriks RGB, pemrosesan gambar selanjutnya dapat dilakukan dengan memanfaatkan berbagai properti yang dimiliki oleh matriks. Pemetaan parameter warna ke dalam vektor dalam ruang warna HSV melibatkan properti matriks seperti perkalian matriks dengan suatu konstanta untuk menormalisasi matriks RGB, dan

langkah-langkah lainnya seperti pada pemecahan masalah untuk ekstraksi fitur warna hingga dihasilkan HSV.

3. Pemetaan citra dalam *co-occurrence matrix* sebagai representasi 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. *Co-occurrence matrix* atau *co-occurrence distribution* merupakan sebuah matriks yang didefinisikan pada suatu gambar sebagai distribusi nilai-nilai piksel yang terjadi bersamaan pada *offset* tertentu. Matriks *co-occurrence* dari suatu gambar mewakili jarak dan hubungan spasial sudut pada subwilayah gambar dengan ukuran tertentu.

4. Ekstraksi fitur tekstur

Pada ekstraksi fitur tekstur, digunakan *co-occurrence matrix* yang selanjutnya juga melibatkan properti matriks dan kalkulasi sedemikian rupa seperti langkah-langkah pada pemecahan masalah hingga diperoleh komponen ekstraksi tekstur (contrast, entropy, homogeneity) sebagai elemen-elemen vektor-fitur tekstur.

5. Penggunaan *cosine similarity* dalam algoritma pencocokan

Penentuan vektor-fitur pada dataset mana yang relevan dengan vektor-fitur pada query gambar pencarian dipandang sebagai pengukuran kesamaan (similarity measure). Semakin sama suatu vektor-fitur dataset dengan vektor-fitur query, semakin relevan dataset tersebut dengan query. Kesamaan (sim) antara dua vektor diukur dengan rumus cosinus similarity yang merupakan bagian dari rumus perkalian titik (dot product) dua buah vektor.

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

III. Ilustrasi kasus dan contoh penyelesaiannya

1. Pemecahan Masalah Warna:

- Ilustrasi:

Seorang pengguna mengunggah gambar berwarna ke dalam website VisuMatch. Pengguna ingin membandingkan warna gambar tersebut dengan dataset gambar yang sudah diunggah sebelumnya dan mencari gambar yang

mirip dengan query gambar yang diberikan.

- Penyelesaian:

- a. Representasikan gambar dalam matriks RGB sebagai representasi warna dari gambar dataset maupun gambar query yang diberikan.

- b. Konversi RGB ke HSV:

Konversi nilai warna dari RGB ke HSV untuk representasi yang lebih sesuai dengan persepsi manusia terhadap warna. Implementasi konversi warna dari RGB ke HSV dilakukan dengan langkah-langkah seperti konversi warna dari RGB ke HSV pada pemecahan masalah.

- c. Histogram Warna:

Hitung histogram warna global HSV untuk gambar input dan setiap gambar dalam dataset. Histogram tersebut merepresentasikan distribusi frekuensi warna dalam gambar.

- d. Cosine Similarity:

Hitung cosine similarity antara histogram warna query gambar dan setiap histogram warna dalam dataset. Hasilnya memberikan tingkat kesamaan warna antara gambar input dan gambar-gambar dalam dataset. Pengukuran kemiripan dari kedua gambar dilakukan dengan menggunakan *cosine similarity* seperti perhitungan *cosine similarity* pada pemecahan masalah. Besar hasil dari *cosine similarity* menunjukkan tingkat kemiripan kedua gambar. Semakin besar nilai *cosine similarity*, maka tingkat kemiripannya semakin tinggi.

2. Pemecahan Masalah Tekstur

- Ilustrasi:

Seorang pengguna ingin mencari gambar dengan tekstur tertentu, misalnya gambar dengan tekstur kasar. Pengguna mengunggah gambar ke dalam *website* VisuMatch dan ingin membandingkan tekstur query gambar tersebut dengan dataset gambar yang sudah diunggah sebelumnya. Sistem akan memberikan hasil berdasarkan kesamaan tekstur.

- Penyelesaian:

- a. Konversi ke Grayscale:

Ubah gambar ke citra grayscale karena fokus pada tekstur tanpa memperhitungkan warna.

b. Kuantifikasi Grayscale:

Lakukan kuantifikasi nilai grayscale untuk mengurangi kompleksitas dan mempersiapkan pembentukan co-occurrence matrix.

c. Co-occurrence Matrix:

Bangun co-occurrence matrix untuk menggambarkan hubungan spasial antara intensitas piksel dalam gambar grayscale.

d. Ekstraksi Komponen Tekstur:

Hitung nilai *contrast*, *entropy*, dan *homogeneity* dari co-occurrence matrix sebagai representasi tekstur. Perhitungan dilakukan sesuai dengan persamaan untuk mendapatkan nilai pada *contrast*, *entropy*, dan *homogeneity* pada pemecahan masalah.

e. Cosine Similarity:

Gunakan cosine similarity untuk membandingkan vektor ekstraksi tekstur gambar input dengan vektor tekstur gambar dalam dataset. Pengukuran kemiripan dari kedua gambar dilakukan dengan menggunakan *cosine similarity* seperti perhitungan *cosine similarity* pada pemecahan masalah. Besar hasil dari *cosine similarity* menunjukkan tingkat kemiripan kedua gambar. Semakin besar nilai *cosine similarity*, maka tingkat kemiripannya semakin tinggi.

3. Implementasi *website*:

- Ilustrasi:

Pengguna mengunjungi *website* VisuMatch dan ingin mengunggah gambar untuk mencari gambar-gambar serupa dalam dataset. Pengguna juga ingin memilih pencarian berdasarkan dua parameter, yaitu warna atau tekstur. Agar lebih memuaskan pengguna ingin *website* dengan tampilan dan fitur yang menarik. Pengguna juga ingin agar dapat menggunakan kamera untuk memasukkan input gambar yang akan dicari kesamaannya.

- Penyelesaian:

Desain antarmuka yang intuitif dengan opsi unggah gambar dan tombol pencarian berdasarkan parameter warna atau tekstur. Antarmuka yang responsif dan ramah pengguna harus diimplementasikan untuk meningkatkan pengalaman pengguna yang lebih baik. *Front-end* akan dibuat sedemikian rupa untuk menyediakan cara bagi pengguna untuk berinteraksi dengan situs

web mencakup segala sesuatu mulai dari mengklik tombol, hingga menggeser atau menyentuh tombol atau fitur lainnya. Akan diimplementasikan juga fitur kamera sehingga pengguna dapat memasukkan input gambar yang akan dicari secara *real-time* dari kamera. Setelah kamera diinisialisasi, program akan mengatur interval waktu di mana penangkapan gambar dilakukan secara otomatis, yaitu setelah 10 detik, program akan mengambil gambar dari webcam. *Website* dibangun dengan antarmuka pengguna yang memungkinkan pengguna untuk mengunggah dataset gambar lalu memasukkan gambar pencarian dan menampilkan hasil berdasarkan kesamaan konten visual dengan parameter warna atau tekstur. Tampilkan gambar-gambar hasil pencarian dengan urutan tingkat kemiripan. Persoalan ini juga berkaitan dengan *pagination* dari pencarian gambar yang dihasilkan. Lalu akan diberikan informasi tambahan seperti tingkat kesamaan atau metrik lainnya.

BAB IV

IMPLEMENTASI DAN UJI COBA

I. Implementasi

1. Ekstraksi fitur warna

- Konversi RGB ke HSV

Citra atau gambar akan direpresentasikan ke dalam matriks RGB. Matriks RGB yang diperoleh dari `image.Image` golang bernilai `uint32` bukan `uint8`. Hal ini membuat normalisasi harus dilakukan terlebih dahulu sebelum dilakukan konversi ke HSV. Normalisasi dilakukan dengan masing-masing elemen matriks RGB dibagi dengan 65535. Konversi dari RGB yang sudah dinormalisasi kemudian dilakukan sesuai dengan langkah-langkah perhitungan yang ada pada pemecahan masalah. Berikut implementasi kodenya dalam pseudocode.

```

function ConvertRGBToHSV(uint32 r, g, b) → schema.HSV
    { Normalize RGB values }
    rf ← r / 65535.0
    gf ← g / 65535.0
    bf ← b / 65535.0

    { Calculate HSV }
    maks ← max(rf, max(gf, bf))
    min ← min(rf, min(gf, bf))
    delta ← maks - min

    { Initialize HSV components }
    h ← 0
    s ← 0
    v ← 0

    { Calculate h }
    if (delta == 0) then
        h ← 0
    else if (maks == rf) then
        h ← 60 * (gf - bf) / delta
    else if (maks == gf) then
        h ← 60 + 60 * (bf - rf) / delta
    else
        h ← 120 + 60 * (rf - gf) / delta

    { Calculate s }
    if (delta == 0) then
        s ← 0
    else
        s ← 255 * delta / maks

    { Calculate v }
    v ← maks

```

```

temp ← (gf - bf) / delta
temp1 ← int(temp) / 6
h ← 60 * (temp - float(temp1 * 6))
else if (maks == gf) then
    h ← 60 * (2 + (bf - rf) / delta)
else if (maks == bf) then
    h ← 60 * (4 + (rf - gf) / delta)

{ Calculate saturation (s) }
if (maks == 0) then
    s ← 0
else
    s ← delta / maks

{ Set v }
v ← maks

{ Create HSV result }
result.H ← h
result.S ← s
result.V ← v

return result

```

- Membuat histogram untuk parameter vektor-fitur warna

Setelah matriks RGB dari representasi gambar dikonversi ke HSV, akan dibuat histogram untuk parameter vektor-fitur warna. Pertama-tama dilakukan create rangemap dan deklarasi setiap part bernilai 0 yang sesuai dengan range masing-masing. Kemudian akan diiterasikan data dan diambil indeks h (hIndex), indeks s (sIndex), dan indeks v (vIndeks) yang sesuai range. Kemudian jika valid indeksnya akan dihitung sesuai dengan perhitungan di bawah sehingga memperoleh nilai indeks array yang akan di increment untuk menjadi nilai histogram. Berikut implementasi kodennya dalam pseudocode.

```

function GetVector(data []schema.HSV) → []int
    { Define HSV ranges }
    hRanges ← [[316.0, 360.0], [1.0, 25], [26.0, 40.0], [41.0, 120.0],
                [121.0, 190.0], [191.0, 270.0], [271.0, 295], [295, 315]]
    sRanges ← [[0, 0.2], [0.2, 0.7], [0.7, 1.01]]
    vRanges ← [[0, 0.2], [0.2, 0.7], [0.7, 1.01]]

    { Initialize histogram with zeros }
    histogram ← array of size 72, filled with zeros

    { Create a map to store counts for each range combination }
    rangeMap ← empty map

    { Initialize counts in the rangeMap to zero }
    for i from 0 to length(hRanges) - 1:
        for j from 0 to length(sRanges) - 1:
            for k from 0 to length(vRanges) - 1:
                rangeMap[i*9 + j*3 + k] ← 0

    { Iterate through each HSV data point }
    for each hsv in data:
        { Initialize indices for H, S, and V }
        hIndex ← -1
        sIndex ← -1
        vIndex ← -1

        { Check the range for H }
        for i, h in enumerate(hRanges):
            if (h[0] <= hsv.H and hsv.H <= h[1]) then
                hIndex ← i
                break

        { Check the range for S }
        for j, s in enumerate(sRanges):

```

```

    if (s[0] <= hsv.S and hsv.S < s[1]) then
        sIndex ← j
        break

    { Check the range for V }
    for k, v in enumerate(vRanges):
        if (v[0] <= hsv.V and hsv.V < v[1]) then
            vIndex ← k
            break

    { If all indices are valid, update the rangeMap }
    if (hIndex != -1 and sIndex != -1 and vIndex != -1) then
        index ← hIndex*9 + sIndex*3 + vIndex
        rangeMap[index]++

    { Copy counts from rangeMap to histogram }
    for index, count in rangeMap.items():
        histogram[index] ← count

return histogram

```

2. Ekstraksi fitur tekstur

- Pemetaan gambar ke representasi matriks *grayscale*

Gambar yang berwarna akan diubah menjadi grayscale karena fokus pada tekstur tanpa memperhitungkan warna. Pengubahan dilakukan sesuai perhitungan rumus pada bagian pemecahan masalah untuk pengubahan RGB ke *grayscale*. Pengubahan dibagi menjadi 16 blok 4 x 4 sehingga membentuk array 3 dimensi. Berikut implementasi kodennya dalam pseudocode.

```

function LoadImageAsGrayscale(string imagePath) → ([][][]uint8, error)
    { Open the image file }
    file, err = os.Open(imagePath)
    if (err is not nil) then
        return nil, err

```

```

defer file.Close()

{ Decode the image }
img, _, err ← image.Decode(file)
if (err is not nil) then
    return nil, err

{ Get image dimensions }
bounds ← img.Bounds()
width, height ← bounds.Max.X, bounds.Max.Y

{ Define block parameters }
block ← 4
h ← array of size (block + 1)
w ← array of size (block + 1)

{ Calculate block boundaries }
for i from 0 to block:
    h[i] ← height * i / block
    w[i] ← width * i / block

{ Initialize variables }
idx ← 0
grayImage ← array of size (block * block)

{ Iterate through each block }
for i from 0 to block:
    for j from 0 to block:
        { Initialize the block in the grayscale image }
        grayImage[idx] ← array of size (h[i + 1] - h[i])
        for y from 0 to (h[i + 1] - h[i]):
            grayImage[idx][y] ← array of size (w[j + 1] - w[j])
            for x from 0 to (w[j + 1] - w[j]):
                { Get the original color at the specified position }

```

```

        originalColor ← img.At(h[i] + x, w[j] + y)
        r, g, b, _ ← originalColor.RGBA()

        { Convert the color to grayscale }
        gray ← 0.299 * float64(r << 8) + 0.587 *
            float64(g << 8) + 0.114 * float64(b << 8)
        grayImage[idx][y][x] ← uint8(gray)

        { Move to the next block }
        idx ← idx + 1

    {Return the grayscale image }
    return grayImage, nil

```

- Membuat *co-occurrence matrix*

Pembuatan *co-occurrence matrix* dilakukan dengan menghitung matriks tiap blok untuk mendapat *co-occurrence*-nya. Hasil yang diperoleh tersebut kemudian dinormalisasi dan jarak euclidean yang dipakai adalah 2 dengan anglenya sebesar 0 derajat. Berikut implementasi kodennya dalam pseudocode.

```

function GetCoOccurrenceMatrix([][]uint8 image) → [][]float64
    { Set quantization level }
    quantization ← 256

    { Initialize the co-occurrence matrix with zeros }
    matrix ← array of size (quantization x quantization)
    for i from 0 to quantization - 1:
        matrix[i] ← array of size quantization
        for j from 0 to quantization - 1:
            matrix[i][j] ← 0

    { Iterate through each pixel in the image }
    for y from 0 to length(image) - 1:
        for x from 0 to length(image[y]) - 3:

```

```

{ Increment co-occurrence count for pairs of pixels separated
  by a distance of 2 }
matrix[image[y][x]][image[y][x + 2]]++
matrix[image[y][x + 2]][image[y][x]]++

{ Normalize the co-occurrence matrix }
size ← 2 * (length(image) * (length(image[0]) - 2))
for i from 0 to quantization - 1:
  for j from 0 to quantization - 1:
    matrix[i][j] ← matrix[i][j] / float64(size)

{ Return the normalized co-occurrence matrix }
return matrix

```

- Menghitung komponen ekstraksi tekstur (*contrast, homogeneity, entropy*)

Penghitungan ekstraksi tekstur meliputi *contrast, homogeneity, entropy* dilakukan sesuai dengan rumus pada pemecahan masalah. Program ini akan menghitung nilai C, H, dan, E yang masing-masing mewakili *contrast, homogeneity, entropy*. Kemudian akan mengembalikan nilai dalam vektor 1 dimensi yang berisi 3 element (ini tiap blok). Berikut implementasi kodenya dalam pseudocode.

```

function GetCHE(matrix [][]float64) → []float64
  { Initialize contrast, homogeneity, and entropy }
  contrast ← 0.0
  homogeneity ← 0.0
  entropy ← 0.0

  { Iterate through each element in the co-occurrence matrix }
  for i from 0 to length(matrix) - 1:
    for j from 0 to length(matrix[i]) - 1:
      { Calculate contrast }
      contrast ← contrast + matrix[i][j] * (i - j) * (i - j)

      { Calculate homogeneity }

```

```

homogeneity ← homogeneity + matrix[i][j] /
(1.0 + (i - j) * (i - j))

{ Calculate entropy }
if (matrix[i][j] > 0) then
    prob ← matrix[i][j]
    entropy ← entropy - prob * log2(prob)

{ Return the contrast, homogeneity, and entropy as an array }
return [contrast, homogeneity, entropy]

```

3. Algoritma pencocokan

- Parameter warna

Algoritma pencocokan untuk CBIR dengan parameter warna dilakukan dengan penghitungan *cosine similarity* sesuai dengan rumus pada pemecahan masalah. Pada program ini, program akan menerima input array of int dan mengembalikan nilai result float64. Berikut implementasi kodennya dalam pseudocode.

```

function CosineSimilarityColor([]int vec1, vec2) → float64
    { Initialize variables for dot product and vector norms }
    dotProduct ← 0.0
    normVec1 ← 0.0
    normVec2 ← 0.0

    { Iterate through each element in the vectors }
    for i from 0 to length(vec1) - 1:
        { Calculate dot product }
        dotProduct ← dotProduct + float64(vec1[i]) * float64(vec2[i])

        { Calculate squared norms of each vector }
        normVec1 ← normVec1 + float64(vec1[i]) * float64(vec1[i])
        normVec2 ← normVec2 + float64(vec2[i]) * float64(vec2[i])

```

```

{ Calculate square root of vector norms }

normVec1 ← sqrt(normVec1)
normVec2 ← sqrt(normVec2)

{ Check if both vectors have non-zero norms }
if (normVec1 > 0 and normVec2 > 0) then
    { Calculate and return cosine similarity }
    return dotProduct / (normVec1 * normVec2)

{ Return 0 for zero norms }
return 0.0

```

- Parameter tekstur

Algoritma pencocokan untuk CBIR dengan parameter tekstur dilakukan dengan penghitungan *cosine similarity* sesuai dengan rumus pada pemecahan masalah. Pada program ini, program akan menerima input array of float64 dan mengembalikan result float64. Berikut implementasi kodennya dalam pseudocode.

```

function CosineSimilarityTexture([ ]float64 vec1, vec2) → float64
    { Initialize variables for dot product and vector norms }
    dotProduct ← 0.0
    normVec1 ← 0.0
    normVec2 ← 0.0

    { Iterate through each element in the vectors }
    for i from 0 to length(vec1) - 1:
        { Calculate dot product }
        dotProduct ← dotProduct + vec1[i] * vec2[i]

        { Calculate squared norms of each vector }
        normVec1 ← normVec1 + vec1[i] * vec1[i]
        normVec2 ← normVec2 + vec2[i] * vec2[i]

    { Check if either vector has a zero norm }

```

```

if (normVec1 == 0 or normVec2 == 0) then
    { Return 0 for zero norms }
    return 0.0

{ Calculate and return cosine similarity }
return dotProduct / (sqrt(normVec1) * sqrt(normVec2))

```

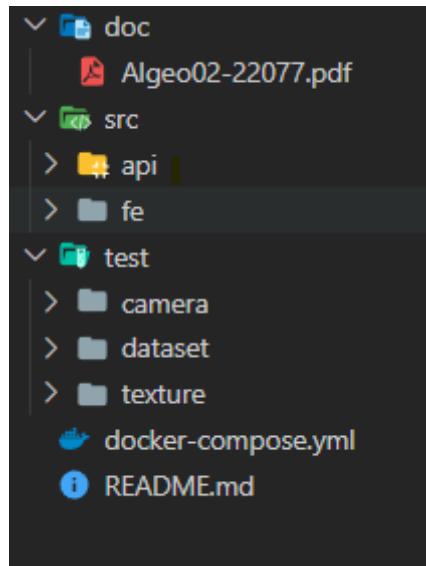
4. Implementasi CBIR

Pengguna akan memasukkan gambar dataset kemudian memasukkan juga query gambar yang akan dicari kesamaannya dalam dataset. Gambar diterima dalam bentuk base64 dan akan divalidasi di setiap form yang ada. Kemudian gambar akan *di-decode* menjadi *image.Image* yang selanjutnya akan diproses. Pemrosesan akan dilakukan dengan mengekstraksi fitur warna dan tekstur dari gambar. Proses ini melibatkan sejumlah langkah-langkah yang telah dijelaskan sebelumnya. Setelah diperoleh vektor-fitur, maka dapat dilakukan pencocokan.

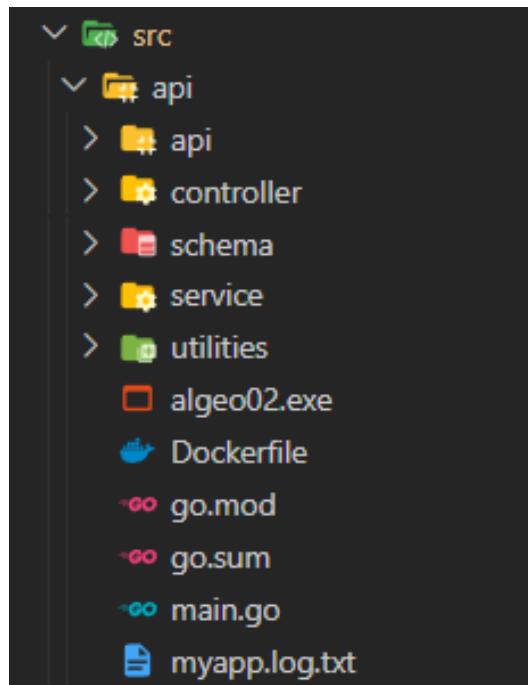
Untuk pencocokan dengan parameter warna akan diakses data vektor-fitur dari *directory* *data/:token/data_color.json*. Sementara untuk pencocokan dengan parameter tekstur akan diakses data vektor-fitur dari *directory* *data/:token/data_texture.json*. json. Data yang diperoleh tersebut akan diubah menjadi map of array vector. Kemudian setiap data map akan dibandingkan dengan menggunakan algoritma pencocokan *cosine similarity*. Hasil dari pencocokan akan disortir dan akan diambil nilai similarity yang lebih dari 60%. Selanjutnya akan dikirimkan hasilnya berupa nama file saja.

Untuk upload dataset, dikirimkan data zip berbentuk base64 kemudian *di-encode* menjadi zipreader. Selanjutnya zipreader akan dilakukan iterasi setiap filenya dan diolah untuk diperoleh hasil vektor setiap gambarnya. File diiterasi kembali untuk disimpan ke lokal dengan thread terpisah agar mempercepat laju program.

II. Struktur program



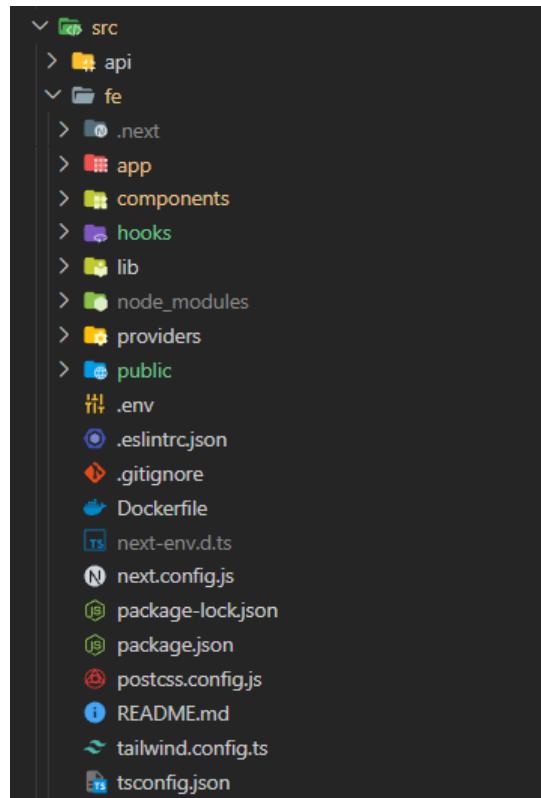
Struktur pada program terbagi menjadi beberapa folder, dimana *source code* program berada pada folder *src*.



Struktur program untuk *back-end* berada pada folder *api*. Digunakan Golang untuk *back-end* ini dengan program utamanya terdapat di *main.go*. Program utama ini digunakan sebagai pusat untuk menyalakan port dan *api*. Di dalam folder *api* terdapat folder *api* lagi dan di dalamnya terdapat file *route.go*. File *route.go* digunakan untuk mengelompokkan endpoint dan deklarasi *url* yang masing-masing memanggil fungsi di *controller*. Ada beberapa endpoint pada file *route.go* ini. Pertama, terdapat endpoint untuk upload dataset (parameter warna saja, tekstur saja, dan keduanya). Lalu ada

endpoint untuk pengecekan warna dan tekstur yang akan di compare dengan data pada json, yaitu vektor-fitur yang disimpan dalam *caching* sesuai dengan token masing-masing *user*. Hal ini memungkinkan program bisa digunakan oleh “*multi-user*” yang masing-masing bisa memiliki dataset yang berbeda. Folder controller berisi *c_color.go*, *c_pagination.go*, dan *c_texture.go*. File *c_color.go* berisikan CBIR dengan parameter warna, sementara *c_texture.go* berisikan CBIR dengan parameter warna tekstur, dan *c_pagination.go* berisikan pagination untuk tampilan hasil pencocokan CBIR. Bagian controller ini akan memanggil beberapa fungsi pembantu di folder utilities. Folder utilities berisikan beberapa fungsi untuk menghitung konversi RGB ke HSV, membuat histogram untuk parameter vektor-fitur warna, dan penghitungan algoritma pencocokan dengan *cosine similarity*. Kemudian folder schema digunakan untuk menyimpan tipe-tipe data bentukan seperti yang diterima dari body request dan untuk mempermudah pengolahan data. Kemudian terdapat dockerfile yang berisikan script untuk setup env docker, di mana env yang disetup adalah env untuk Chrome (dalam perealisasian *scrapping* yang menggunakan *headless browser*) kemudian setup untuk keperluan golang.

GIN dipilih sebagai framework utama untuk membangun API backend. Dengan sintaksis yang sederhana dan performa yang tinggi, GIN memungkinkan kami untuk dengan mudah membuat endpoint-endpoint API yang responsif dan dapat berskala. Docker digunakan untuk kontainerisasi aplikasi sehingga dapat berjalan dengan mudah di berbagai komputer tanpa perlu setup yang rumit. Dengan menggunakan Docker, kami dapat mengisolasi aplikasi dan dependensinya, memastikan portabilitas dan konsistensi lingkungan di berbagai sistem operasi. Untuk ekstraksi RGB dari gambar digunakan `image.Image` dari GoLang. Hal ini dapat mengintegrasikan kemampuan pemrosesan gambar secara langsung ke dalam lingkungan GoLang. Chrome DP memberikan solusi efektif untuk scraping gambar yang bersifat dinamis, memanfaatkan Chromium untuk rendering halaman web dan mengekstrak data. Sampai memungkinkan kami melakukan scraping gambar berdasarkan teks, membuka peluang untuk mencari dan mendapatkan gambar dengan hasil yang lebih relevan. Yandex memberikan opsi tambahan untuk scraping gambar berdasarkan teks. Hal ini dilakukan dengan mengimplementasikannya melalui chrome DP sehingga dapat mengakses hasil scraping secara dinamis.



Di dalam folder src juga terdapat folder fe. Folder fe ini berisi semua kode yang berhubungan dengan *front-end* dari *website* yang kami buat. Untuk frontend kami menggunakan framework next.js sehingga untuk struktur programnya juga mengikuti struktur program dari next.js itu sendiri. Folder app merupakan folder utama dalam program ini. Di dalam folder app ini berisi routing dari website yang kami buat, dan implementasi semua page yang ada di website kami terletak pada folder ini. Selanjutnya terdapat folder components yang berisi komponen-komponen *react* yang digunakan untuk mendukung dan membangun tiap *page* dari website kami. Folder lib berisi fungsi-fungsi pendukung yang dibutuhkan dalam *website* kami seperti konversi dari file ke base64, zip file, dan lain-lain. Folder hooks berisi custom react hooks yang diperlukan dalam *website* kami. Folder provider berisi provider-provider yang diperlukan untuk membangun website kami seperti provider untuk toaster dan juga provider untuk library *user interface* yang kami pakai, yaitu nextUI. Folder public berisi asset-asset static (foto) yang digunakan dalam website kami seperti foto kami pribadi untuk halaman about us dan aset gambar lainnya. Folder .next sebenarnya adalah folder hasil dari build next.js itu sendiri. Lalu terdapat folder node_modules berisi data-data dependency yang kami perlukan untuk membangun website kami.

React.js merupakan sebuah library javascript yang digunakan untuk membangun *front-end* dari sebuah *website*. Library ini menggabungkan HTML langsung dengan javascript sehingga akan memudahkan kami dalam proses pembuatan website yang interaktif dan responsif. Seperti yang sudah kami jelaskan sebelumnya kami memilih menggunakan library ini karena menurut kami akan lebih mudah membangun website dengan bantuan react dibandingkan harus menggunakan vanilla html, css, dan javascript.

Next.js merupakan sebuah framework react yang digunakan untuk mempermudah membangun sebuah website terutama dari sisi *front-end*-nya. Framework ini kami pilih karena kami sendiri lebih terbiasa menggunakan framework ini dibandingkan dengan framework lainnya. Framework ini juga mempermudah masalah routing yang ada di sebuah *website*. Selain itu, dengan adanya fitur *server side rendering* pada framework ini dapat meningkatkan performa dari *website* VisuMatch yang dibuat.

Tailwind CSS merupakan framework CSS yang digunakan untuk mempermudah dan mempercepat pembuatan aplikasi dengan design custom. Framework ini menggunakan prinsip *inline styling* dimana kami bisa mengatur *style* dari suatu komponen hanya dengan satu baris saja. Kami memilih framework ini karena kami merasa bahwa framework ini dapat mempermudah dan mempercepat proses pembangunan website kami.

NextUI merupakan sebuah framework *user interface* untuk react dengan berbagai *pre-build* dan *pre-style* komponen. Kami menggunakan framework ini untuk mempercepat proses pembangunan website kami. Framework ini kami gunakan untuk membuat komponen seperti *button*, *pagination*, *navbar*, dan komponen lainnya.

III. Penggunaan program

Secara keseluruhan *website* kami terdiri dari 5 *page* yaitu *home page*, *description page*, *search default page*, *search camera page*, dan yang terakhir *about us page*.

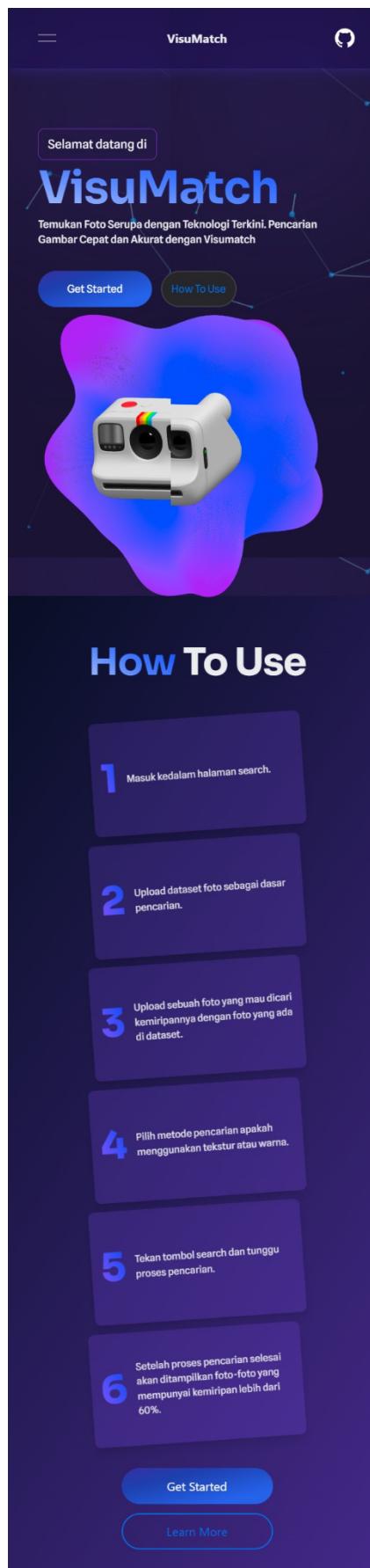
1. Home page

Home page berisi pengenalan singkat mengenai *website* kami dan juga cara menggunakan website kami secara singkat.

- Tampilan desktop



- Tampilan mobile



2. Description page

Description page berisi penjelasan singkat dari algoritma CBIR yang kami gunakan, penjelasan cara pemakaian *website* kami secara lebih lengkap, dan juga teknologi apa saja yang kami gunakan untuk membangun *website* kami.

- Tampilan desktop

The screenshot shows the VisuMatch website interface. At the top, there is a navigation bar with links for Home, Description (which is the active page), About, and Search. The main content area has a dark blue background with white text and icons.

Description:

Search image yang kami buat menerapkan teori CBIR atau Content-Based Image Retrieval. Content-Based Image Retrieval (CBIR) merupakan sebuah proses yang digunakan untuk mencari dan mengambil gambar berdasarkan konten visual gambar seperti warna, tekstur, dan bentuk, tanpa bergantung pada kata kunci atau teks. Hal ini membuat CBIR sangat berguna dalam mengakses dan mengeksplorasi koleksi gambar ketika tidak ada informasi teks atau kata kunci yang cukup untuk mendeskripsikan gambar yang dicari. Fitur-fitur penting dari gambar seperti warna, tekstur, dan bentuk, akan diekstraksi terlebih dahulu sebelum selanjutnya akan diwakili dalam bentuk vektor atau deskripsi numerik. Lalu CBIR menggunakan algoritma pencocokan untuk membandingkan vektor-fitur dari gambar yang dicari dengan vektor-fitur gambar dalam dataset. Hasil dari pencocokan selanjutnya dapat digunakan untuk mengurutkan gambar-gambar dalam dataset dan menampilkan gambar yang paling sesuai dengan gambar yang dicari. Implementasi CBIR yang paling populer digunakan adalah CBIR dengan parameter warna dan CBIR dengan parameter tekstur.

How To Use:

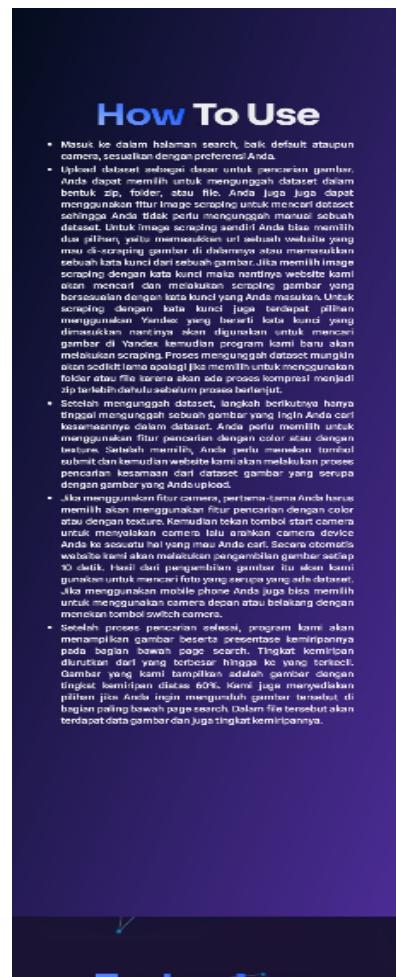
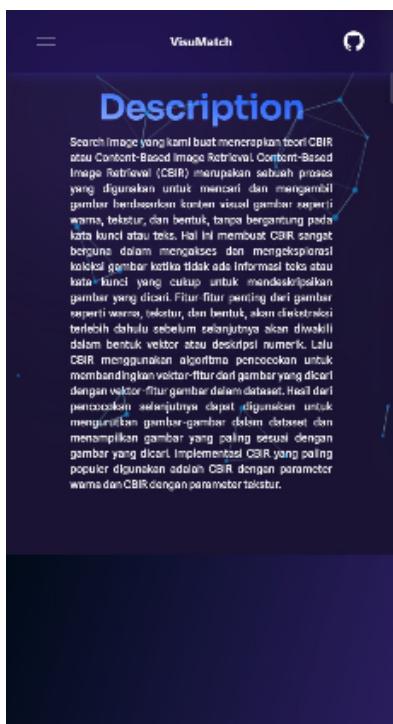
- Masuk ke dalam halaman search, baik default ataupun camera, sesuaikan dengan preferensi Anda.
- Upload dataset sebagai dasar untuk pencarian gambar. Anda dapat memilih untuk mengunggah dataset dalam bentuk zip, folder, atau file. Anda juga dapat menggunakan fitur image scraping untuk mencari dataset sehingga Anda tidak perlu mengunggah manual sebuah dataset. Untuk image scraping sendiri Anda bisa memilih dua pilihan, yaitu memasukkan url sebuah website yang mau di-scraping gambar di dalamnya atau memasukkan sebuah kata kunci dari sebuah gambar. Jika memilih image scraping dengan kata kunci maka nantinya website kami akan mencari dan melakukan scraping gambar yang bersesuaian dengan kata kunci yang Anda masukan. Untuk scraping dengan kata kunci juga terdapat pilihan menggunakan Yandex yang berarti kata kunci yang dimasukkan nantinya akan digunakan untuk mencari gambar di Yandex kemudian program kami baru akan melakukan scraping. Proses mengunggah dataset mungkin akan sedikit lama apalagi jika memilih untuk menggunakan folder atau file karena akan ada proses kompresi menjadi zip terlebih dahulu sebelum proses berlanjut.
- Setelah mengunggah dataset, langkah berikutnya hanya tinggal mengunggah sebuah gambar yang ingin Anda cari kesamaannya dalam dataset. Anda perlu memilih untuk menggunakan fitur pencarian dengan color atau dengan texture. Setelah memilih, Anda perlu menekan tombol submit dan kemudian website kami akan melakukan proses pencarian kesamaan dari dataset gambar yang serupa dengan gambar yang Anda upload.
- Jika menggunakan fitur camera, pertama-tama Anda harus memilih akan menggunakan fitur pencarian dengan color atau dengan texture. Kemudian tekan tombol start camera untuk menyalaikan camera lalu arahkan camera device Anda ke sesuatu hal yang mau Anda cari. Secara otomatis website kami akan melakukan pengambilan gambar setiap 10 detik. Hasil dari pengambilan gambar itu akan kami gunakan untuk mencari foto yang serupa yang ada dataset. Jika menggunakan mobile phone Anda juga bisa memilih untuk menggunakan camera depan atau belakang dengan menekan tombol switch camera.
- Setelah proses pencarian selesai, program kami akan menampilkan gambar beserta presentase kemiripannya pada bagian bawah page search. Tingkat kemiripan diurutkan dari yang terbesar hingga ke yang terkecil. Gambar yang kami tampilkan adalah gambar dengan tingkat kemiripan diatas 60%. Kami juga menyediakan pilihan jika Anda ingin mengunduh gambar tersebut di bagian paling bawah page search. Dalam file tersebut akan terdapat data gambar dan juga tingkat kemiripannya.

Technology:

The technology section displays eight logos in a grid:

- N (Node.js)
- Atom (Atom Editor)
- =GO (GO Language)
- TS (TypeScript)
- Docker (Docker Container)
- Node.js (Node.js logo)
- React Native (React Native logo)
- Chrome (Google Chrome browser logo)

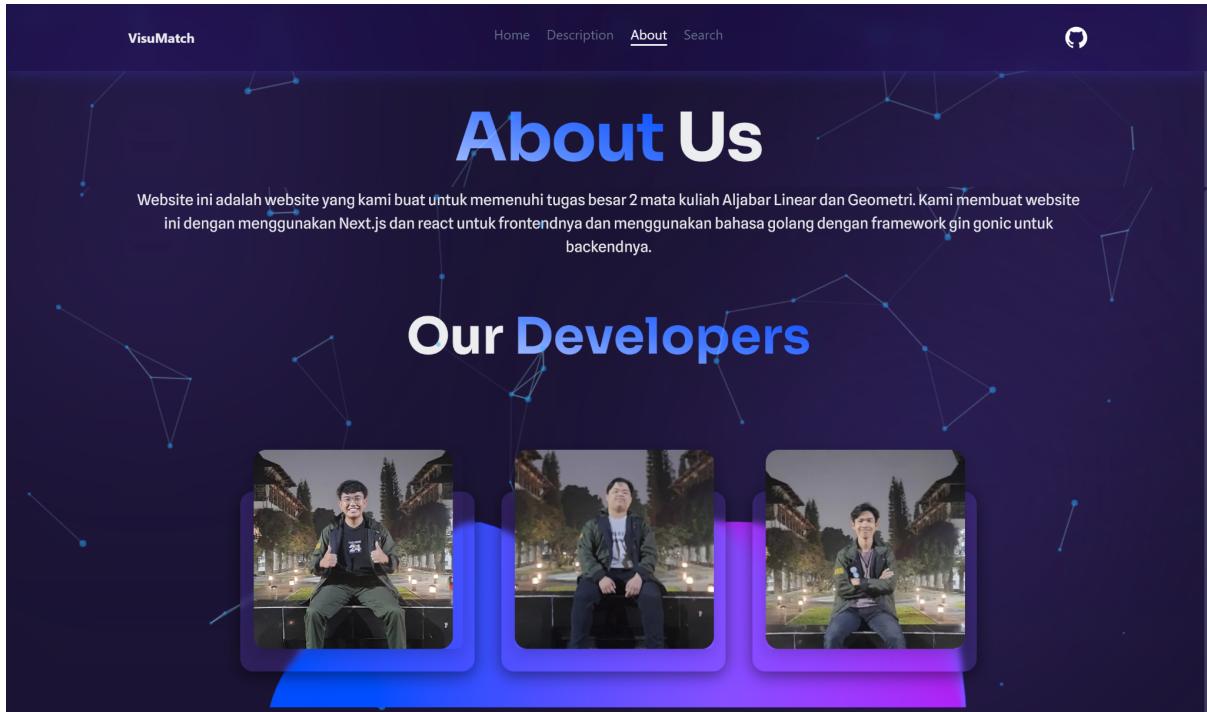
- Tampilan mobile



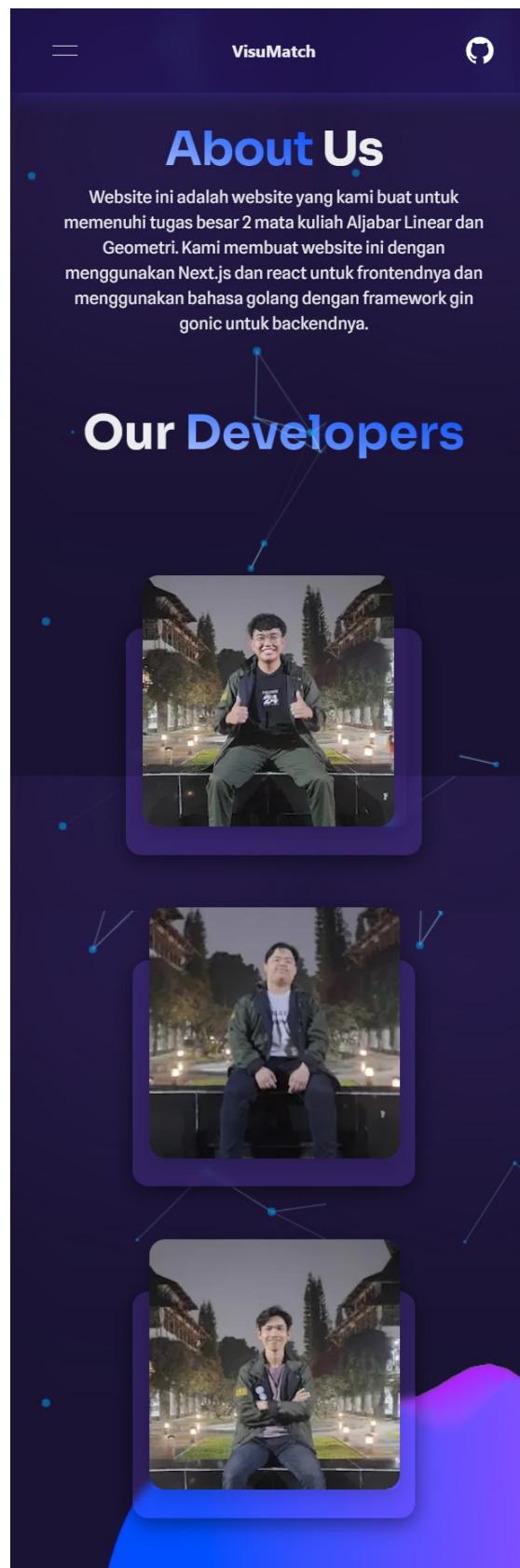
3. About us page

Halaman ini berisi tentang data diri dari pembuatan *website* yaitu kami sendiri.

- Tampilan desktop



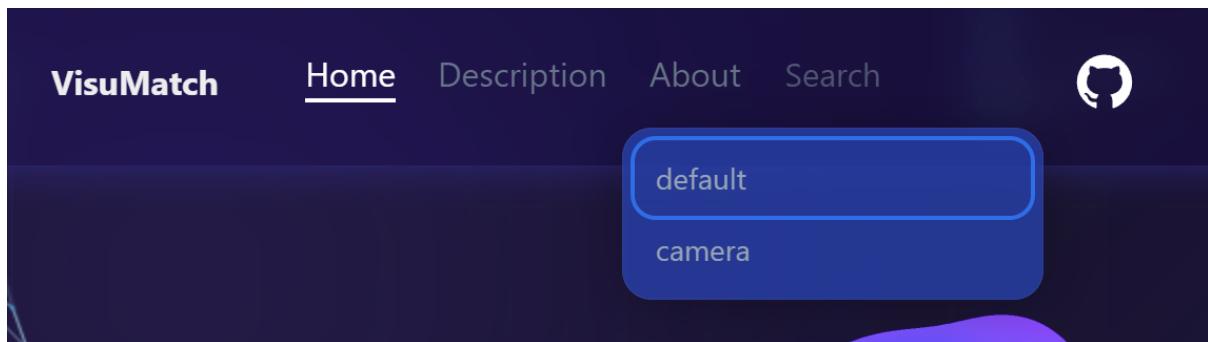
- Tampilan mobile



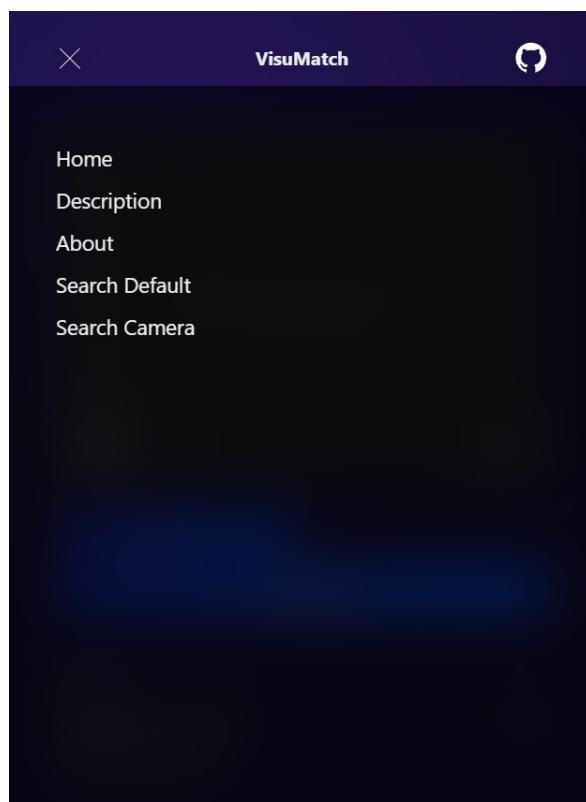
4. *Search*:

Pada menu *search*, pengguna dapat melakukan pencarian kesamaan gambar menggunakan dua metode untuk memasukkan gambar, yaitu secara default mengunggah gambar dan secara *real-time* menggunakan *camera*.

- Tampilan desktop



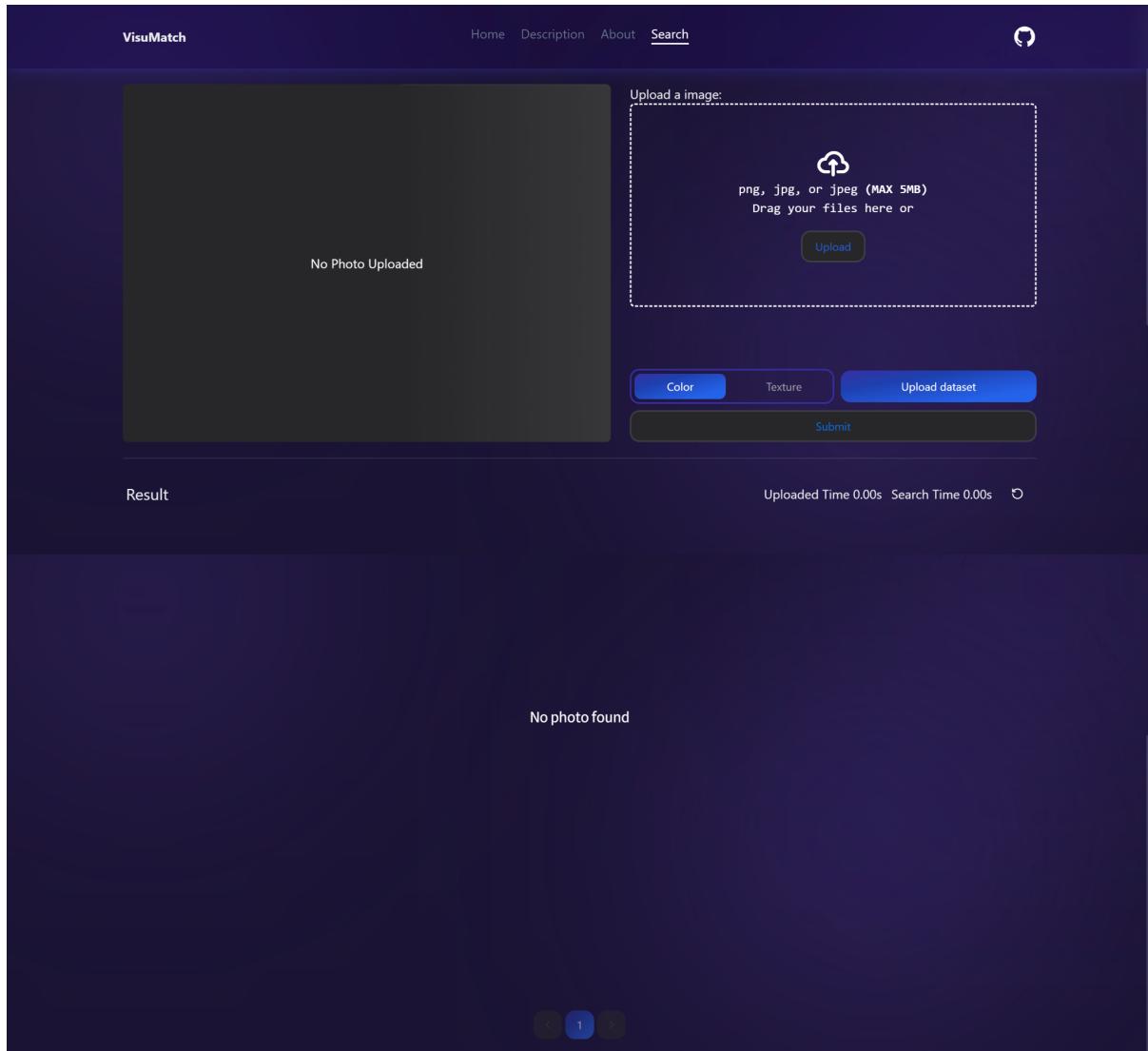
- Tampilan mobile



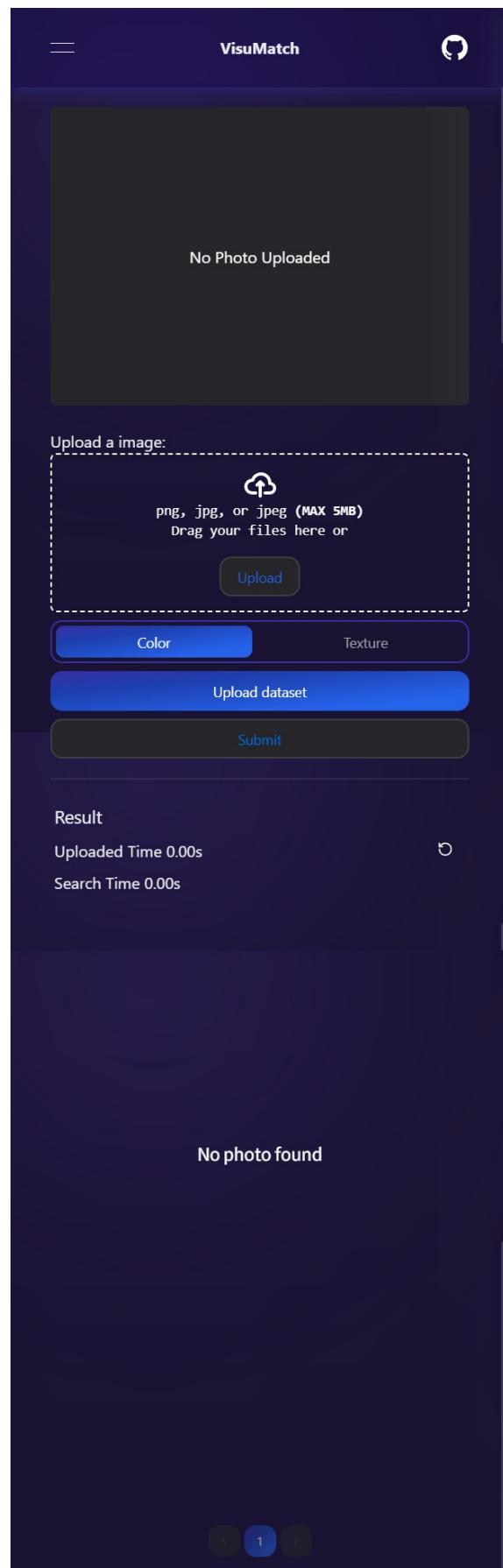
a. *Search default page*

Halaman ini merupakan halaman utama dari *website* kami dimana halaman ini merupakan halaman dimana pengguna akan melakukan pencarian foto yang serupa.

- Tampilan desktop



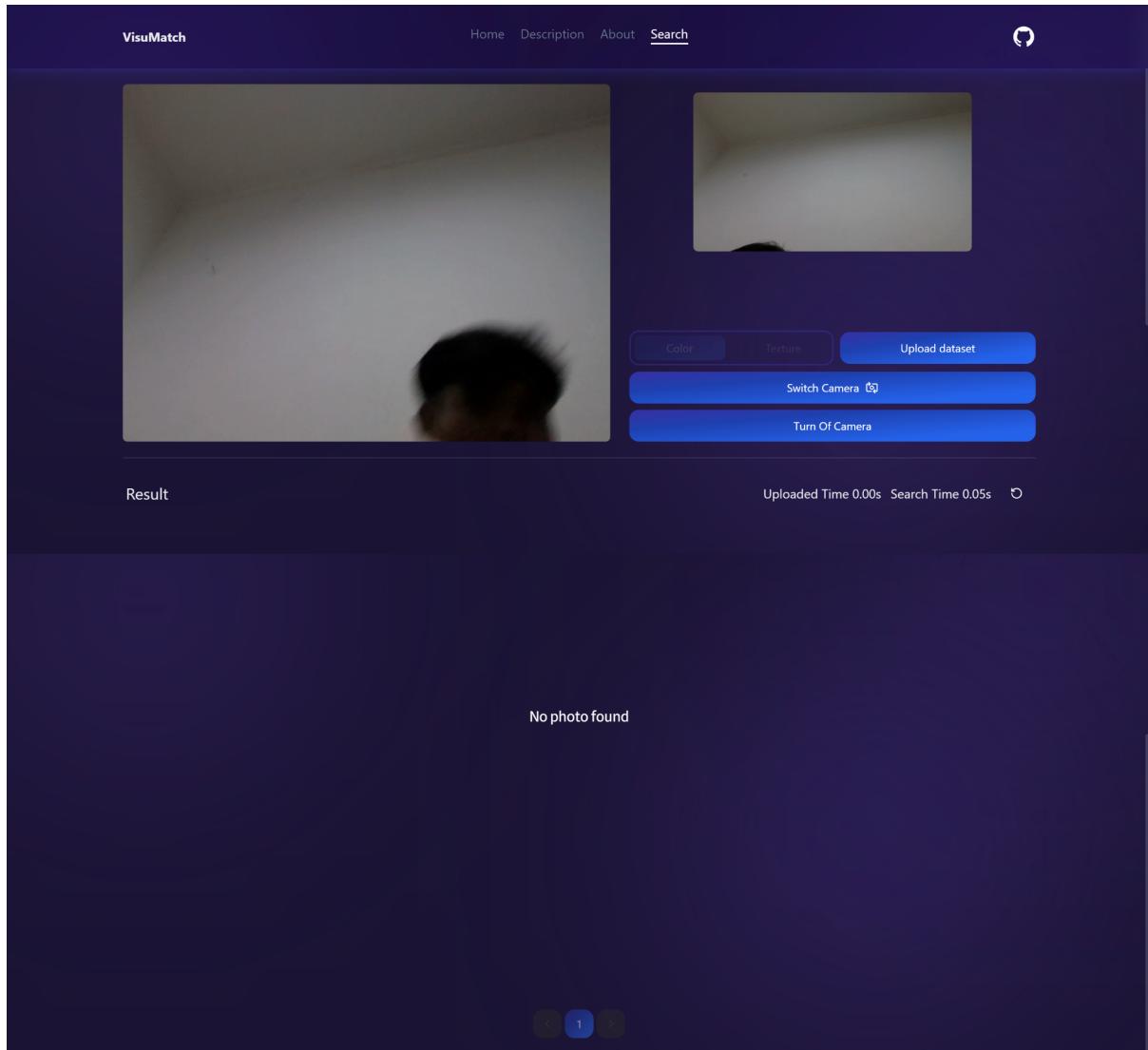
- Tampilan mobile



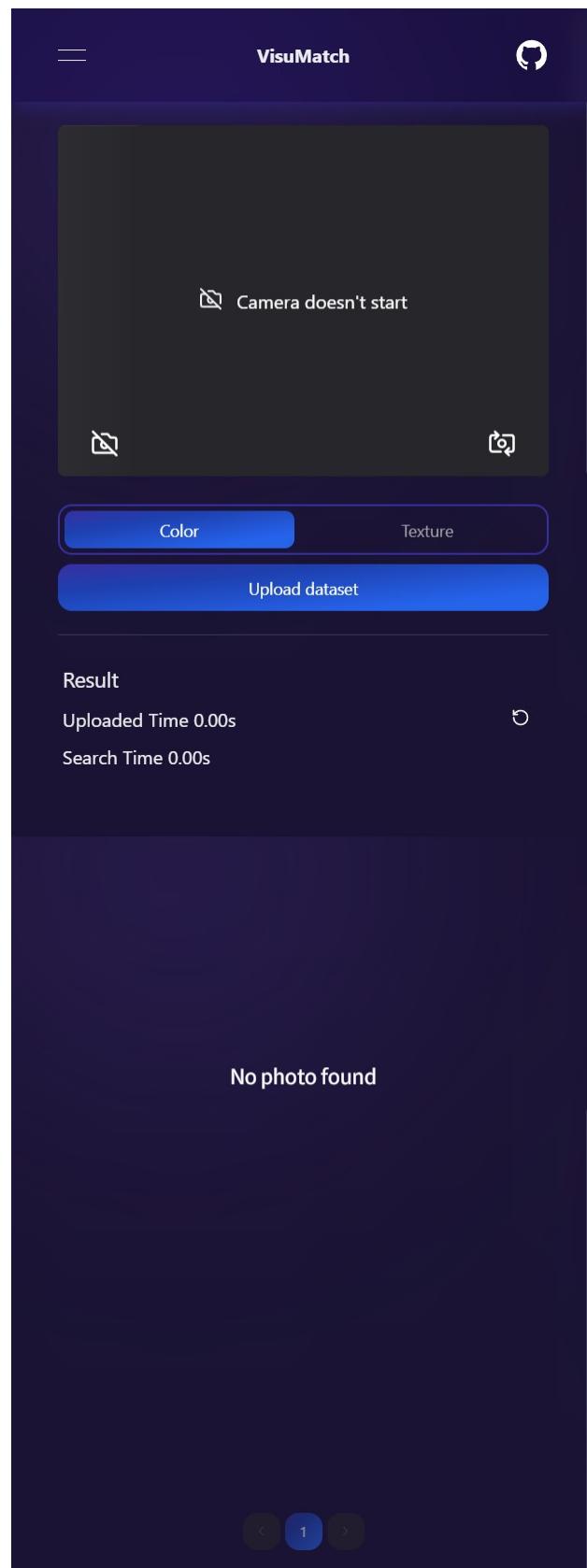
b. *Search camera page*

Halaman ini tidak jauh berbeda dengan *search default page* hanya saja pada halaman ini gambar yang dicari tidak diupload pengguna namun langsung ditangkap dengan camera setiap 10 detik.

- Tampilan desktop



- Tampilan mobile



Untuk menggunakan fitur *search* pada program kami langkah pertama yang perlu dilakukan pastilah harus masuk kedalam search page terlebih dahulu. Untuk masuk ke dalam search page bisa melalui tombol yang ada di *navigasi bar* atau tombol “*get started*” yang ada di *home page*.

Setelah masuk kedalam *search page* langkah berikutnya adalah mengunggah dataset dengan cara menekan tombol *upload* dataset. Untuk mengunggah dataset kami menyediakan beberapa pilihan, Anda bisa memilih mengunggah dalam bentuk zip, folder, maupun dalam bentuk files. Selain itu, kami juga menyediakan fitur *image scraping* untuk mengumpulkan dataset. Untuk fitur *image scraping* sendiri kami menyediakan 3 pilihan. Pilihan yang pertama adalah memasukan url dari website yang ingin di-*scraping* (pastikan *url* yang dimasukkan harus sesuai dengan format yang ada). Pilihan yang kedua adalah memasukkan kata kunci dari foto yang ingin dijadikan dataset. Sebagai contoh jika memasukan kata “apel”, maka *website* kami akan melakukan pencarian gambar “apel” yang kemudian akan kami *scraping* dan kami jadikan dataset. Untuk pilihan ketiga sebenarnya sama dengan pilihan kedua hanya saja sumber gambar pada pilihan ketiga ini akan berasal dari yandex image. Proses mengunggah gambar mungkin akan sedikit lebih lama jika memilih untuk mengunggah folder atau files karena terdapat poses zip file terlebih dahulu.

Setelah mengupload dataset langkah yang berikutnya adalah mengupload sebuah query gambar atau gambar yang akan dicari kesamaannya pada dataset. Hal ini dilakukan dengan cara menekan tombol *upload* atau melakukan *drag and drop* gambar pada kotak yang sudah disediakan. Setelah gambar diunggah, langkah berikutnya adalah memilih metode untuk pencarian yaitu *color* atau *texture* pada *toggle button* yang sudah disediakan. Setelah memilih metode pencarian langkah berikutnya hanya perlu menekan tombol *submit* dan menunggu proses pencarian.

Jika menggunakan fitur kamera, langkah pertama yang dilakukan tidak jauh berbeda yaitu mengunggah dataset dan memilih metode pencarian terlebih dahulu. Setelah melakukan hal tersebut langkah berikutnya adalah mengaktifkan kamera dengan cara menekan tombol *start camera*. Ketika kamera sudah menyala maka program kami akan secara otomatis melakukan pengambilan gambar setiap 10 detik dan setelah itu langsung melakukan pencarian. Jika menggunakan *mobile phone* juga terdapat tombol *switch camera* untuk mengganti dari kamera depan ke kamera belakang atau sebaliknya.

Setelah proses pencarian selesai, gambar dengan tingkat kemiripan di atas 60% akan ditampilkan secara terurut pada bagian bawah *page search*. Ketika sebuah foto di-*hover* akan ditampilkan tingkat kemiripannya dan juga terdapat pilihan untuk mengunduh gambar tersebut. Waktu eksekusi untuk mengunggah file dan juga waktu eksekusi untuk *search* juga akan ditampilkan di bagian tengah halaman. Di bagian bawah halaman juga terdapat pilihan untuk mengunduh semua hasil pencarian ke dalam bentuk pdf. Di dalam file pdf tersebut akan terdapat data gambar beserta tingkat kemiripannya.

IV. Hasil pengujian

1. Menggunakan dataset yang diberikan pada spesifikasi (kumpulan gambar hewan), akan digunakan 500 foto pertama saja, menggunakan metode pencarian dengan warna.

- Tes 1:

The screenshot shows the VisuMatch search interface. At the top, there is a navigation bar with links for Home, Description, About, and Search. The Search link is underlined, indicating it is the active page. To the right of the navigation is a user icon. Below the navigation is a search input field with the placeholder "Upload an image:" and instructions: "png, jpg, or jpeg (MAX 5MB) Drag your files here or". There is also a "Upload" button. Below the search input is a row of three buttons: "Color", "Texture", and "Upload dataset". A "Submit" button is located below this row. On the left side of the main area is a large image of a tiger's face. In the center, there is a grid of eight smaller images showing various animals: two tigers, two lions, two jaguars, and two cheetahs. Below the grid is a navigation bar with arrows and page numbers (1, 2, 3, 4). At the bottom of the page is a "Download PDF" button.

Image with similarity 100.000000



Image with similarity 71.134419



Image with similarity 69.527917

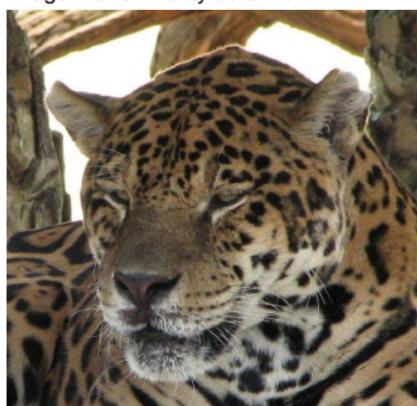


Image with similarity 67.898375



Hasil pencarian secara lengkap untuk tes 1 dengan parameter warna ini terdapat pada link berikut:

https://drive.google.com/file/d/1RRL7ovZC4Qae3L7wYScRAV5MyhNJxi5_/view?usp=sharing

- Tes 2:

The screenshot shows the VisuMatch search interface. At the top, there is a navigation bar with links for Home, Description, About, and Search. Below the navigation bar is a search input field labeled "Upload a image:" with a placeholder "png, jpg, or jpeg (MAX 5MB) Drag your files here or". There is also a "Upload" button. Below the search input are three buttons: "Color", "Texture", and "Upload dataset". A "Submit" button is located below these buttons. On the left side of the interface, there is a large image of a snow leopard's head and shoulders. Below the search input, the word "Result" is displayed. To the right of "Result", it says "Uploaded Time 12.11s Search Time 0.18s" and there is a refresh icon. Below this, there are three smaller images of snow leopards, each showing a different view of the animal's face and upper body. At the bottom of the interface, there are navigation arrows for a page number indicator (1) and a "Download PDF" button.

Image with similarity 100.000000



Image with similarity 60.253589

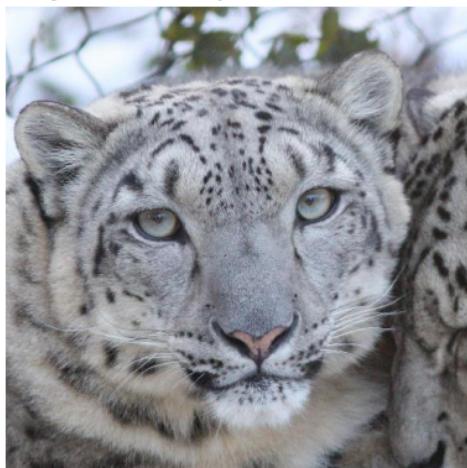
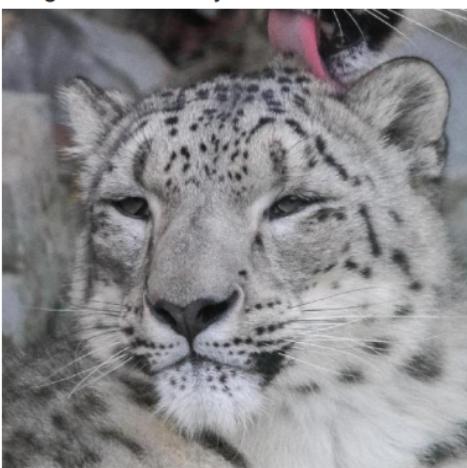


Image with similarity 60.198781



Hasil pencarian secara lengkap untuk tes 2 dengan parameter warna ini dapat dilihat pada link berikut:

<https://drive.google.com/file/d/1GodOCOL5a6GyYw77MgZ8xFOO6Ri-sT5M/view?usp=sharing>

2. Menggunakan dataset texture yang ada di folder test kami, menggunakan metode pencarian dengan texture.

- Tes 1:

The screenshot shows the VisuMatch search interface. At the top, there is a navigation bar with links for Home, Description, About, and Search (which is underlined). To the right of the navigation is a user icon. Below the navigation is a search input field with a placeholder "Upload an image:" and a "Upload" button. Below the search input are three buttons: Color, Texture (which is highlighted in blue), and Upload dataset. A "Submit" button is located below these buttons. The main area is titled "Result" and displays a grid of images. The first image in the grid is a black silhouette of a duck, which matches the query image. Other images include a deer silhouette, a landscape with a mountain, a tree in fog, a person with an umbrella crossing a street, a color calibration chart, and a geometric pattern chart. At the bottom of the grid are navigation arrows and a "Download PDF" button.

Image with similarity 100.000000



Image with similarity 87.837685



Image with similarity 87.509144

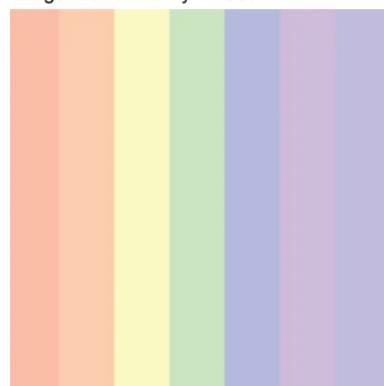


Image with similarity 87.508840



Hasil pencarian secara lengkap untuk tes 1 dengan parameter tekstur ini dapat dilihat pada link berikut:

<https://drive.google.com/file/d/1hRN66XPfC-eUj1dG4czs0fUB9hIg1-yO/view?usp=sharing>

- Tes 2:

The screenshot shows the VisuMatch search interface. At the top, there is a navigation bar with links for Home, Description, About, and Search. A GitHub icon is also present. Below the navigation bar, there is a large input field labeled "Upload a image:" with a dashed border. Inside the field, there is a cloud icon and the text "png, jpg, or jpeg (MAX 5MB) Drag your files here or". Below the input field is a "Upload" button. Underneath the input field, there are three buttons: "Color", "Texture" (which is highlighted in blue), and "Upload dataset". Below these buttons is a "Submit" button. To the left of the input field, there is a large, vibrant abstract image with a radial burst effect of red, blue, green, yellow, and white colors. On the right side of the interface, there is a "Result" section. It displays a grid of eight images. The first row contains four images: a colorful abstract image, a person with a red umbrella crossing a wet street, a vertical color calibration bar, and another vertical color calibration bar. The second row contains four images: a tree in a foggy landscape, a 3x3 grid of geometric patterns (hexagons, crosses, etc.), a woman wearing a hat, and a black and white photo of a dog. At the bottom of the result section, there is a navigation bar with arrows and page numbers (1, 2, 3). At the very bottom, there is a "Download PDF" button.

Image with similarity 100.000000



Image with similarity 99.999993

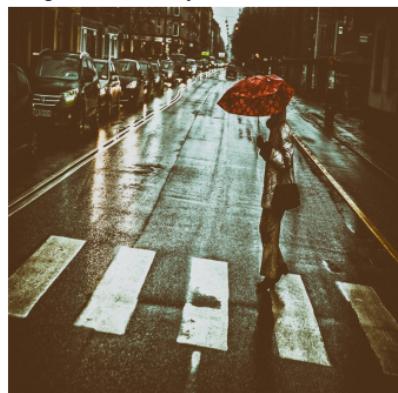


Image with similarity 99.999988

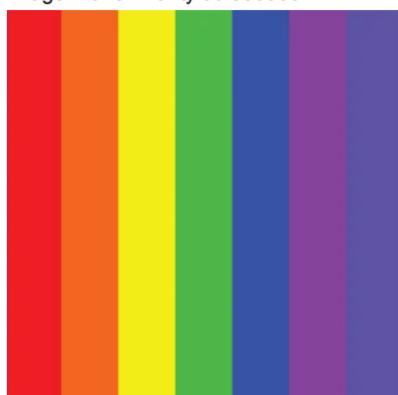


Image with similarity 99.999971



Hasil pencarian secara lengkap untuk tes 2 dengan parameter tekstur ini dapat dilihat pada link berikut:

https://drive.google.com/file/d/1prJlWlQmNYAqfUK0IySIdaS5jm_HvJ-X/view?usp=sharing

3. Dataset akan menggunakan fitur image scraping yandex dengan kata kunci apel, dan metode pencarian yang digunakan adalah color.

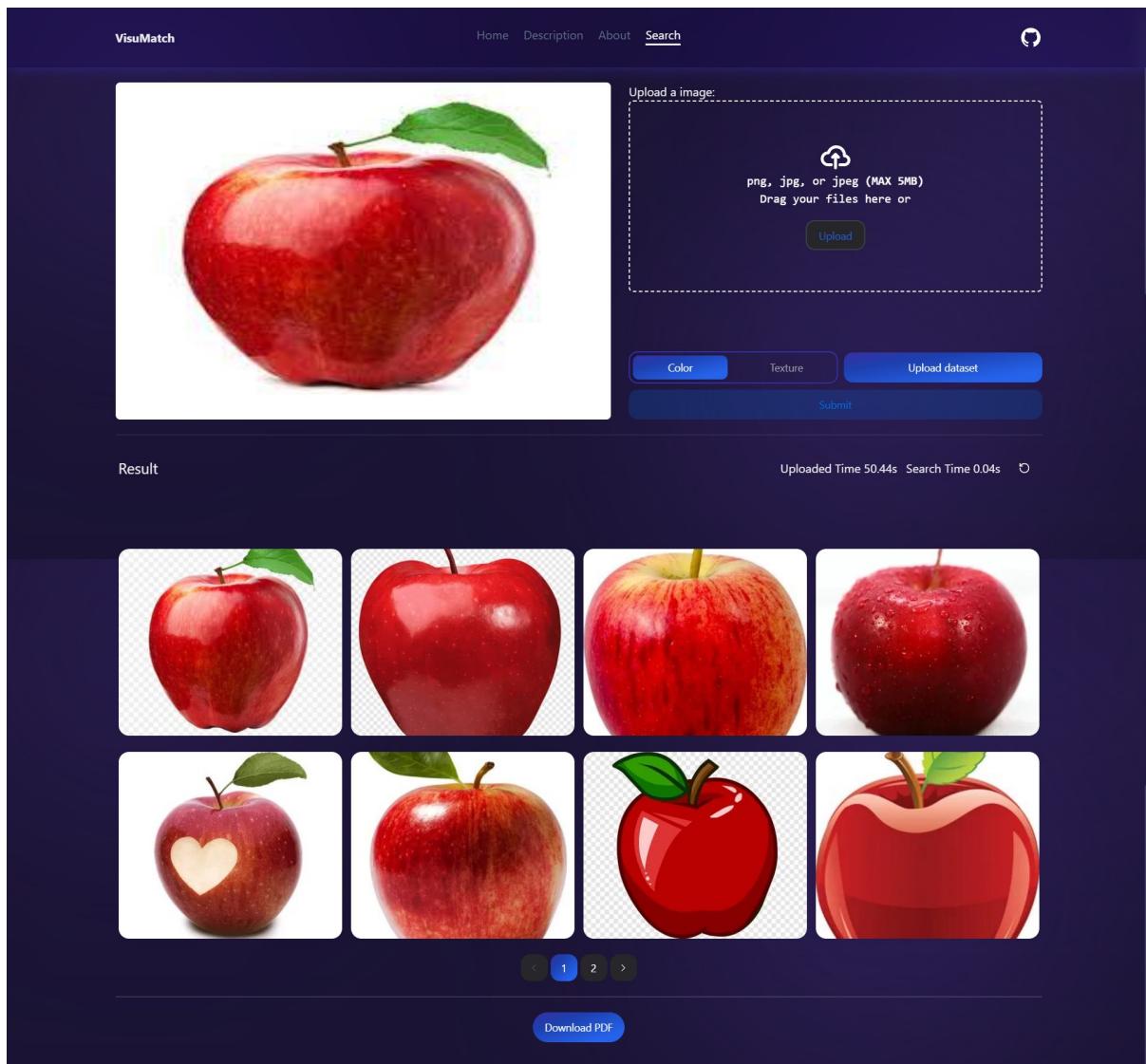


Image with similarity 95.166444

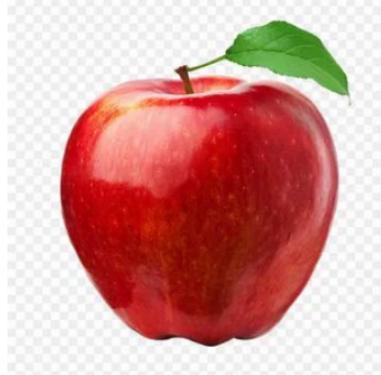


Image with similarity 71.471379

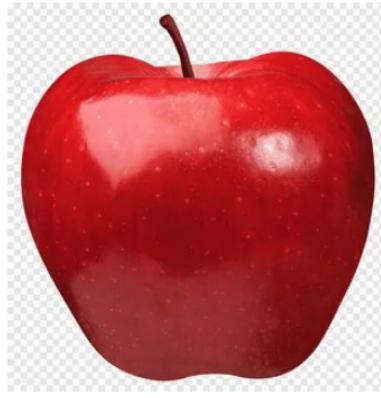
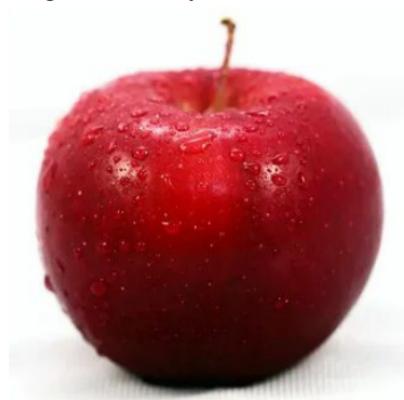


Image with similarity 69.521858



Image with similarity 68.982067



Hasil pencarian secara lengkap untuk tes ini dapat dilihat pada link berikut:

[https://drive.google.com/file/d/1thW8zm4Y8KuztgS_kKs2tHxXKjLirQh5/vie
w?usp=sharing](https://drive.google.com/file/d/1thW8zm4Y8KuztgS_kKs2tHxXKjLirQh5/view?usp=sharing)

4. Dataset menggunakan web scrap url pada link url berikut:

“<https://scrapeme.live/product-category/pokemon>”.

Metode pencarian dengan menggunakan color.

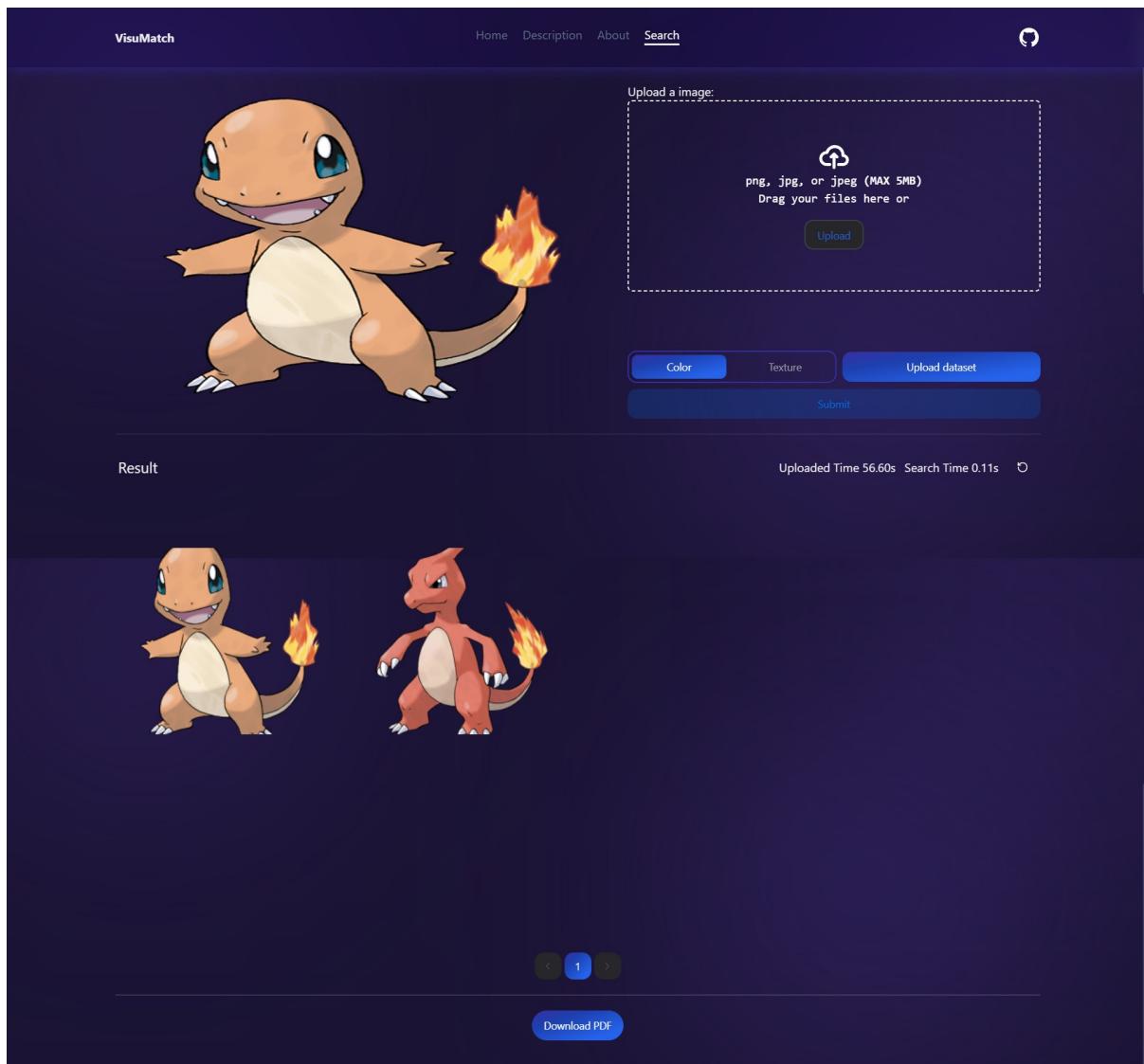


Image with similarity 89.469117



Image with similarity 70.045777



Hasil pencarian lengkap dapat dilihat pada link berikut:

<https://drive.google.com/file/d/1l3YHwIkolub03eBmX5uAqLd00PzS6jdd/view?usp=sharing>

5. Dataset menggunakan image scraping text dengan kata kunci “*low contrast image*”, Metode pencarian menggunakan tekstur.

VisuMatch

Home Description About Search

Upload a image:

png, jpg, or jpeg (MAX 5MB)
Drag your files here or

Upload

Color Texture Upload dataset

Submit

Result

Uploaded Time 53.21s Search Time 0.08s

The screenshot shows the VisuMatch search interface. At the top, there's a navigation bar with links for Home, Description, About, and Search. Below the navigation is a large input field labeled "Upload a image:" with a "Upload" button. To the right of the input field are three buttons: "Color", "Texture" (which is highlighted in blue), and "Upload dataset". Below these buttons is a "Submit" button. The main area is titled "Result" and displays a grid of 13 images. The first image in the grid is a low-contrast photograph of a forest. The other 12 images are higher-contrast versions of various scenes, including a woman's face, a field of flowers, a sunset over water, two girls, a patterned fabric, trees, and a beach. Some of these images have labels like "shutterstock.com - 2142174799", "Cross Exposure For Contrast Control", "Cross Exposure For Dark And Light Areas", "Without Filter", and "With Filter". At the bottom of the result grid is a navigation bar with page numbers from 1 to 13, with "1" being the active page. Below the grid is a "Download PDF" button.

shutterstock.com - 2142174799

Cross Exposure For Contrast Control

Cross Exposure For Dark And Light Areas

Without Filter

With Filter

Download PDF

Image with similarity 99.999875



Image with similarity 99.999783



Image with similarity 99.999777

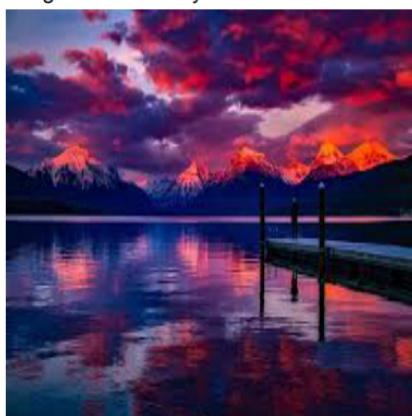


Image with similarity 99.999773



Hasil pencarian lengkap dapat dilihat pada link berikut:

[https://drive.google.com/file/d/1-yKi36jAUFi55pl74GFUPSggS8Q10pHG/vie
w?usp=sharing](https://drive.google.com/file/d/1-yKi36jAUFi55pl74GFUPSggS8Q10pHG/view?usp=sharing)

6. Dataset menggunakan folder camera pada folder test yang ada pada susunan folder kami. Metode pencarian menggunakan color, dan akan digunakan fitur camera.

- Tes 1:

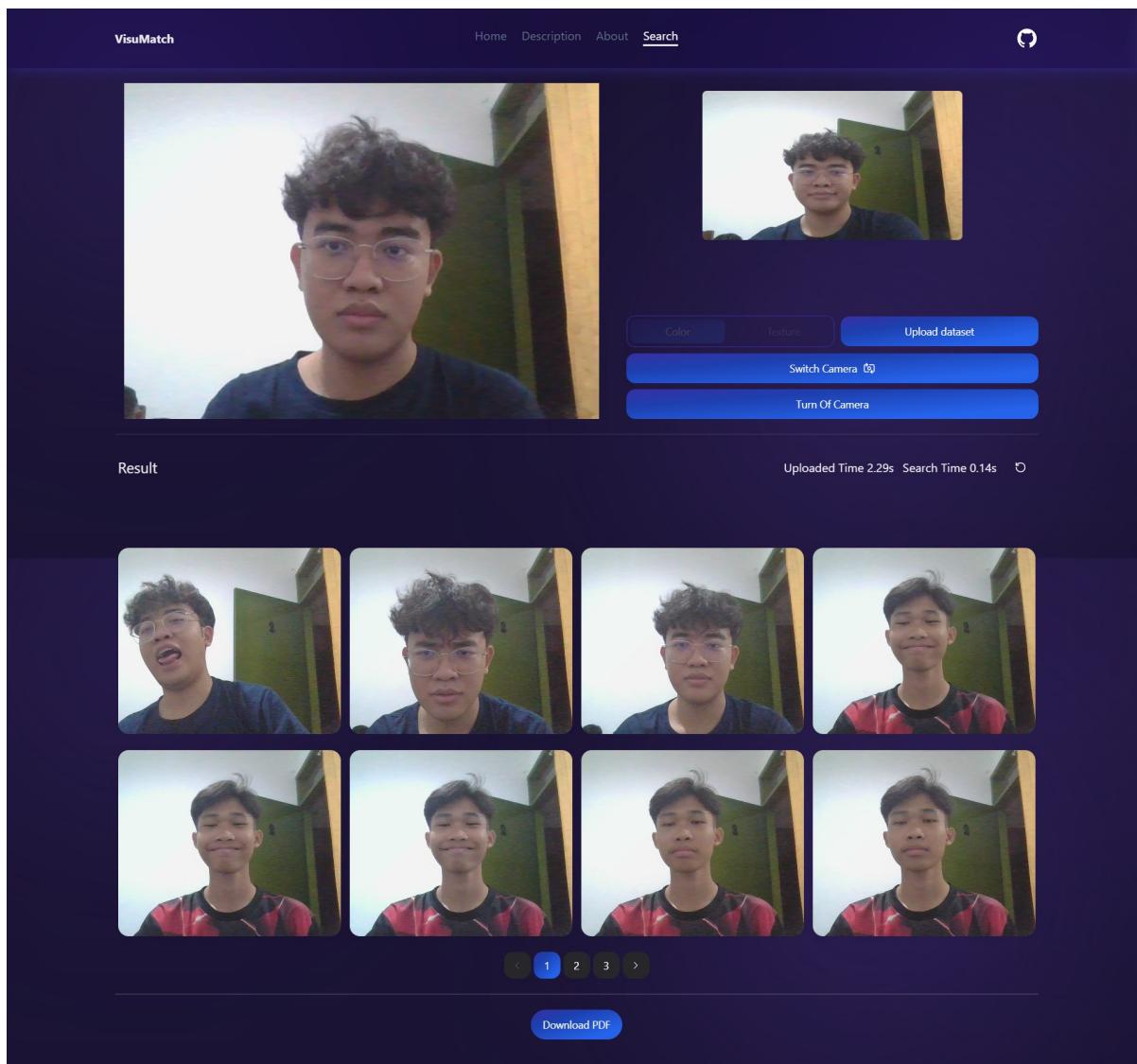


Image with similarity 88.437444



Image with similarity 82.995241



Image with similarity 81.790311



Image with similarity 78.247769



Hasil pencarian secara lengkap dapat dilihat pada link berikut:

<https://drive.google.com/file/d/1vXIXtv7W9eOBjeeUUiJDaA1cUX3hzXo0/view?usp=sharing>

- Tes 2:

The screenshot shows the VisuMatch web application interface. At the top, there is a navigation bar with links for Home, Description, About, and Search. Below the navigation bar, there are two input fields: one for Color and one for Texture. A GitHub icon is located in the top right corner.

The main area features a large input field containing a query image of a man's face. To the right of the input field are three buttons: "Upload dataset", "Switch Camera" (with a camera icon), and "Turn Of Camera".

Below the input field, the text "Result" is displayed, followed by the search statistics: "Uploaded Time 2.29s" and "Search Time 0.12s".

The search results are presented in a grid format. The first row contains four images of the same man from different angles. The second row contains four more images of the same man, also from different angles. Below the grid is a navigation bar with page numbers 1, 2, 3, 4, and arrows for navigating through the results.

At the bottom of the results section, there is a blue button labeled "Download PDF".

Image with similarity 83.796720



Image with similarity 83.643468

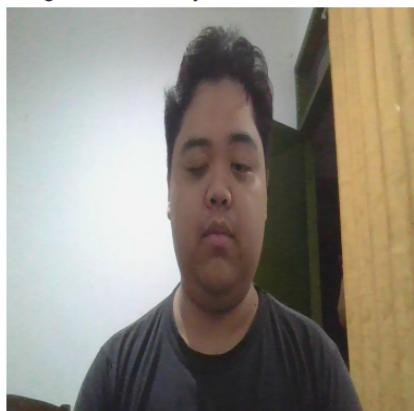


Image with similarity 83.280347

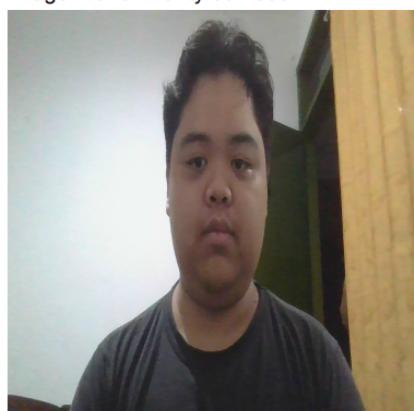


Image with similarity 82.613738



Hasil pencarian secara lengkap dapat dilihat pada link berikut:

<https://drive.google.com/file/d/1yndD4bsMzSdA1aD5ympsid3G0Dt1iO2O/view?usp=sharing>

- Tes 3: dengan metode pencarian tekstur

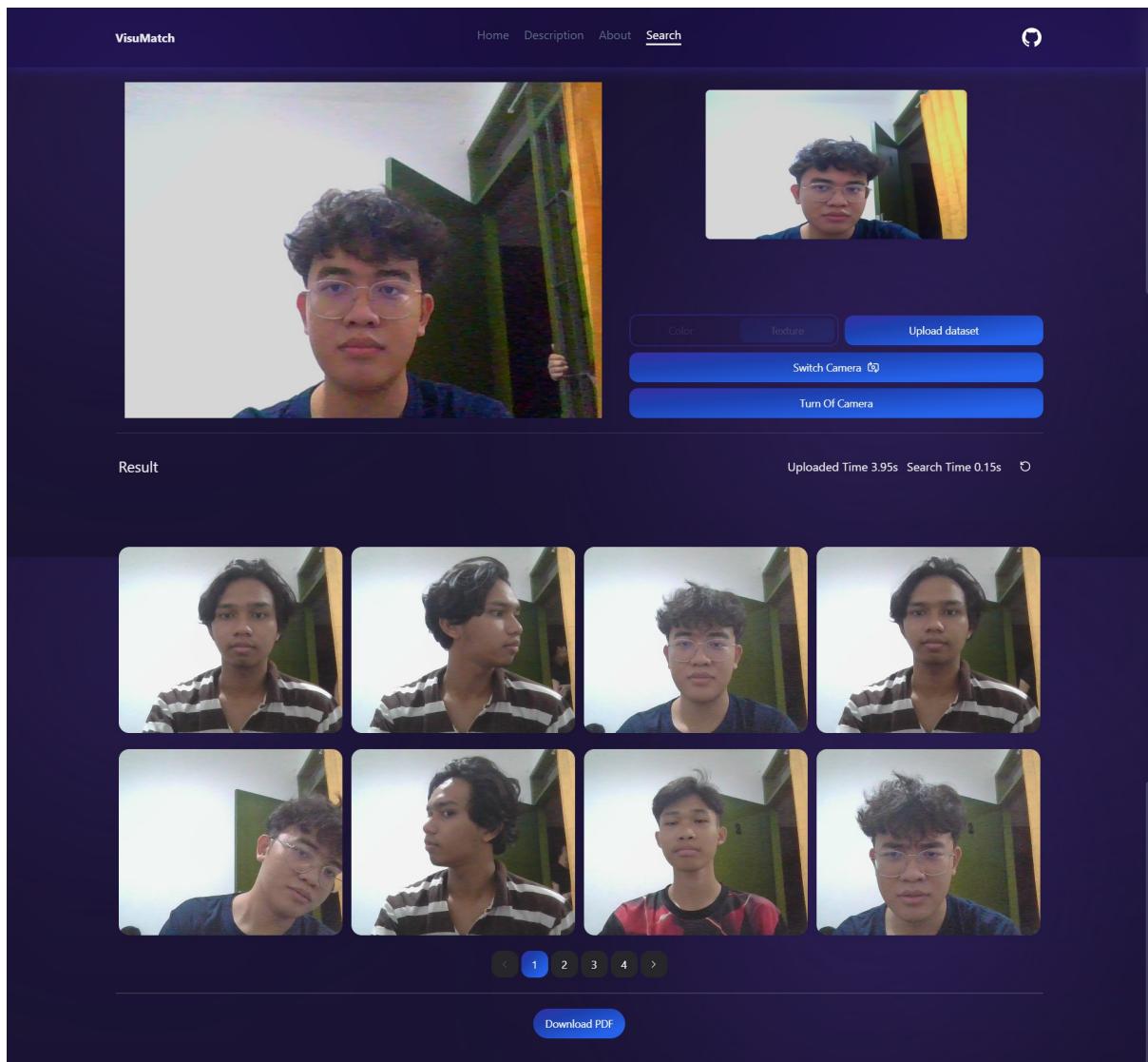


Image with similarity 93.756491



Image with similarity 93.756053



Image with similarity 93.755253



Image with similarity 93.754995



Hasil pencarian secara lengkap dapat dilihat pada link berikut:

<https://drive.google.com/file/d/1TAtLy8645l8DOP4UoaYZajq8J5hSjPEn/view?usp=sharing>

V. Analisis desain solusi algoritma CBIR

Setelah dilakukan pengujian (*testing*), *Content-Based Image Retrieval* (CBIR) menggunakan warna seringkali memberikan hasil yang lebih akurat dibandingkan dengan CBIR yang berfokus pada tekstur. Hal ini terjadi karena jumlah elemen vektor yang lebih banyak dalam CBIR warna, yaitu 72 elemen. Semakin banyaknya elemen untuk perbandingan ini yang memungkinkan perhitungan yang lebih tepat daripada CBIR tekstur yang hanya memiliki 3 elemen pada setiap vektornya.

Dalam kedua metode CBIR tersebut, sebuah foto dibagi menjadi 16 blok. Blok yang berada di tengah-tengah memiliki bobot dua, sedangkan blok di sekitarnya memiliki bobot satu. Hasil perhitungan didasarkan pada total kesamaan *cosine similarity* yang dibagi dengan 20. Oleh karena itu, akurasi dari CBIR ini akan meningkat apabila konten utama dari gambar yang ingin dibandingkan terletak di tengah-tengah.

Untuk meningkatkan kecepatan pemrosesan gambar dan mengubahnya menjadi histogram, kami menggunakan go routine dari bahasa pemrograman Go. Pendekatan ini memungkinkan proses setiap iterasi berjalan secara paralel dengan batas maksimal 130 thread, yang signifikan dalam mempercepat pemrosesan gambar. Selanjutnya, dalam proses upload dataset, kami menerapkan konsep yang serupa dengan "optimistic update". Hal ini dilakukan melalui serangkaian proses penyimpanan file dan pengolahan gambar menjadi vektor sebelum akhirnya akan disimpan ke dalam format JSON, dan proses ini dilakukan secara terpisah. Pendekatan ini mampu membuat program berjalan 10 kali lebih cepat dari metode konvensional.

BAB V

PENUTUP

I. Kesimpulan

Dalam tugas besar ini, kami berhasil mengimplementasikan sistem temu balik gambar (*image retrieval system*) berbasis *Content-Based Image Retrieval* (CBIR) dengan memanfaatkan aljabar vektor. Program dapat mencari dan mengelompokkan gambar berdasarkan kesamaan nilai citra visual, baik dari segi warna atau pun tekstur. Pemanfaatan aljabar vektor memberikan pendekatan klasifikasi berbasis konten yang efektif dalam memproses dan mengevaluasi kemiripan gambar. Dari implementasi program diperoleh bahwa *Content-Based Image Retrieval* (CBIR) menggunakan warna seringkali memberikan hasil yang lebih akurat dibandingkan dengan CBIR yang berfokus pada tekstur. Hal ini terjadi karena jumlah elemen vektor yang lebih banyak dalam CBIR warna, yaitu 72 elemen. Semakin banyaknya elemen untuk perbandingan ini yang memungkinkan perhitungan yang lebih tepat daripada CBIR tekstur yang hanya memiliki 3 elemen pada setiap vektornya.

II. Saran

Untuk meningkatkan kecepatan dalam pencarian kesamaan gambar perlu dilakukan optimasi lebih lanjut pada algoritma pencocokan gambar. Hal ini melibatkan peningkatan efisiensi perhitungan, terutama pada dataset yang berjumlah sangat-sangat besar, agar program dapat memberikan respons yang lebih cepat. Penambahan dokumentasi pada program juga diperlukan sehingga akan lebih memudahkan lagi dalam proses debugging. Selain itu, untuk meningkatkan kecepatan dalam generate PDF dibutuhkan library yang mampu menangani 16 bit *depth image* sehingga tidak perlu melakukan konversi ke 8 bit *depth image*, yang mana hal ini akan mempersingkat waktu dalam generate PDF.

III. Komentar atau tanggapan

Penyelesaian tugas ini memberikan kontribusi yang mendalam untuk meningkatkan pemahaman kami terkait materi kuliah IF2123 - Aljabar Linier dan Geometri, khususnya pada penerapan algoritma pencocokan menggunakan cosine similarity serta pengenalan pada aplikasi dalam bidang pengolahan citra. Melalui penggerjaan tugas ini, kami dapat merasakan pengalaman eksplorasi yang

menyenangkan, mengingat kami menikmati setiap tahap pembuatan aplikasi berbasis *website* ini dari awal hingga akhir.

IV. Refleksi

Kami menyadari bahwa keberjalanannya dalam menyelesaikan tugas besar ini membutuhkan kerjasama yang erat dan komunikasi yang efektif di antara anggota kelompok kami. Kami telah melakukan koordinasi yang baik sejak awal, mulai dari bonding untuk membuat kerja sama kami menjadi lebih baik, pembagian tugas hingga penentuan batas waktu untuk setiap progress yang dikerjakan. Diskusi bersama secara aktif juga menjadi langkah kunci untuk menyelesaikan berbagai permasalahan yang muncul.

Selain itu, time management juga sangat penting untuk diterapkan mengingat jadwal akademik yang padat dan tiap individu yang memiliki urusan masing-masing dan kepentingan yang berbeda. Penting untuk tetap mengutamakan keseimbangan antara tugas akademis dan kehidupan pribadi, termasuk aspek-aspek seperti tidur yang mempengaruhi kesejahteraan fisik dan mental. Kami sadar bahwa manajemen waktu berdampak pada kualitas tidur kami, sehingga kami berusaha menerapkan manajemen waktu yang efektif. Hal ini membuat kami merasa senang dan menikmati proses penggerjaan tugas besar ini.

Dengan rasa syukur, kami menyelesaikan tugas besar ini yang telah menjadi fokus kami selama dua minggu terakhir. Proses penggerjaan ini menjadi salah satu pengalaman paling memuaskan di antara semua tugas besar lainnya yang pernah kami hadapi. Kami ingin menyampaikan terima kasih kepada Enrique, Naufal, Mesach, dan Bapak Judi Santoso, serta semua asisten Aljabar Linier dan Geometri Tahun 2022 atas pengalaman tubes yang sangat menyenangkan ini.

V. Ruang perbaikan atau pengembangan

Terkait pengembangan di masa depan, terdapat beberapa aspek yang dapat diperbaiki atau dikembangkan. Menambahkan fitur ekstensibilitas untuk mendukung penggunaan berbagai jenis dataset dan memudahkan penggunaan program di berbagai konteks. Melibatkan teknik *machine learning* untuk meningkatkan akurasi dalam pemilihan gambar yang mirip. Penggunaan model *machine learning* dapat memberikan hasil yang lebih canggih dan akurat. Meningkatkan interaksi pengguna

dengan menambahkan fitur pencarian lanjutan, filter, atau kategori untuk memudahkan pengguna dalam navigasi dan mengelola dataset gambar. Untuk meningkatkan akurasi dari program ini, konsep *image detector* akan memberi dampak yang baik karena dapat meningkatkan akurasi. Selain itu, untuk meningkatkan pengalaman pengguna dan personalisasi dapat dibuat fitur log in dalam mengimplementasikan sistem yang *multi-user*.

DAFTAR PUSTAKA

- Anton, Howard. Elementary Linear Algebra, 10th edition, John Wiley and Sons, 2010.
- Chrome DP Documentation. Diakses dari <https://pkg.go.dev/github.com/chromedp/chromedp>.
- Fairweather, Lewis. Scrape the Web Faster, in Go with Chromedp. Diakses dari <https://itnext.io/scrape-the-web-faster-in-go-with-chromedp-c94e43f116ce> pada 15 November 2023.
- Manandar, Geshan. How to use Next.js with Docker and Docker compose a beginner's guide. Diakses dari <https://geshan.com.np/blog/2023/01/nextjs-docker/> pada 10 November 2023.
- McGowan, Lee. Write a Lightweight API and Static File Server in Go. Diakses dari <https://medium.com/swlh/write-a-lightweight-api-and-static-file-server-in-go-5e5b208ccdaf> pada 6 November 2023.
- Munir, Rinaldi. IF2123 Aljabar Geometri - Semester I Tahun 2023/2024. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/> pada 8 November 2023.
- Yunus, M. Feature Extraction : Gray Level Co-occurrence Matrix (GLCM). Diakses dari <https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1> pada 13 November 2023.

LAMPIRAN**Link Repositories:**

<https://github.com/mybajwk/Algeo02-22077.git>