LAPORAN TUGAS BESAR 2
IF2123 ALJABAR LINEAR DAN GEOMETRI

Aplikasi Aljabar Vektor dalam Sistem Temu Balik Gambar

Disusun oleh:

Ignatius Jhon Hezkiel Chan	(13522029)
Suthasoma Mahardhika Munthe	(13522098)
Marvin Scifo Y. Hutahaean	(13522110)

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

2023

Daftar Isi

BAB I
Deskripsi Masalah

1. Tujuan

Tujuan pengerjaan tugas besar ini adalah:

1.1. Mempelajari dan mengenali *web development* dengan membuat website sebagai tempat untuk mengeksekusi sebuah aplikasi

1.2. Membuat aplikasi yang menerima masukan dalam bentuk gambar dan menampilkan kecocokan dari gambar tersebut dengan dataset yang dipunya serta menampilkan waktu eksekusi dari program tersebut

2. Penjelasan Masalah

Pada masa ini, banyak sekali gambar-gambar yang disimpan di dalam database internet. Biasanya orang menggunakan *search engine* seperti Google dan memasukkan *keyword* untuk mendapatkan apa yang diinginkan pengguna. Namun, terkadang ada beberapa pengguna yang tidak mengetahui kata yang baik untuk mendeskripsikan apa yang ingin dicari sehingga ketika memasukkan *keyword* yang dia pikir adalah yang tercocok, gambar atau website yang dicari biasanya tidak sesuai yang dia inginkan. Hal ini disebabkan karena barang yang ingin dicari bukanlah berdasarkan kata-kata melainkan berdasarkan gambar dan dia tidak bisa mendeskripsikan gambar tersebut dengan baik untuk dimasukkan ke *keyword*. Perlu ada sebuah fitur yang *search engine*-nya bisa menerima gambar dan mencari kesamaan gambar tersebut dengan data-data yang ada.

Search engine yang bisa menerima masukkan berupa gambar biasanya bisa ditemukan di berbagai situs dan contohnya adalah Google Lens. Google Lens menerapkan sistem yang bernama Sistem Temu Balik Gambar atau Content Based Image Retrieval (CBIR). Sistem temu balik gambar ini bekerja dengan mengidentifikasi gambar berdasarkan konten visual, seperti warna dan tekstur. CBIR adalah bentuk dari aplikasi Aljabar Vektor yang digunakan untuk menggambarkan dan menganalisis data. Dengan metode CBIR ini, ada 2 parameter yang digunakan yaitu dengan parameter warna dan parameter tekstur. Dari semua ini, program CBIR diimplementasikan ke dalam Back-End dari sebuah website. Hal ini dilakukan sebagai salah satu komponen dari sebuah website *Reverse Image Search*.

BAB II
Landasan Teori

1. **Dasar-Dasar Teori 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. Pada Tugas Besar kali ini, Anda diminta untuk mengimplementasikan 2 parameter CBIR yang paling populer, antara lain :

1. CBIR dengan parameter warna

Pada CBIR kali ini akan dibandingkan *input* dari sebuah *image* 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.

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

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

2. Mencari *Cmax*, *Cmin*, dan Δ

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

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

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

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

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

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

$$V = Cmaks$$

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* × *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 disarankan menggunakan 4 × 4 blok.

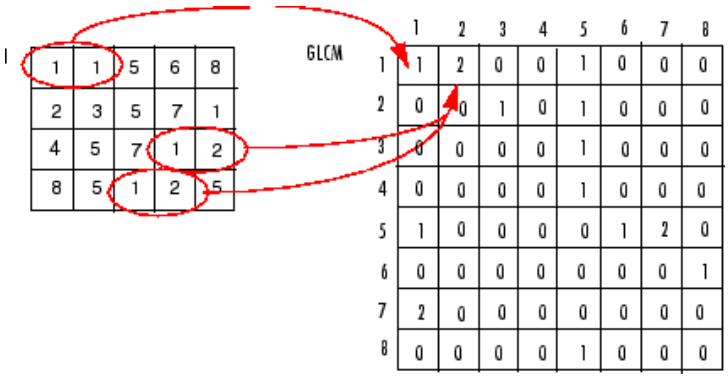
2. CBIR dengan parameter tekstur

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* × *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.

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

Melalui persamaan tersebut, digunakan nilai θ adalah 0°, 45°, 90°, dan 135°. Sebagai gambaran, berikut diberikan contoh cara pembuatan *co-occurrence matrix* dan link cara pembuatannya (Contoh ini dapat digunakan sebagai referensi, bukan acuan sebagai acuan utama).



Gambar 2. Cara Pembuatan Matrix *Occurrence*

Setelah didapat *co-occurrence matrix*, buatlah *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 :

1. 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:

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

2. 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
3. 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.

4. 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.

2. Dasar-Dasar Pengembangan Web

Pengembangan Web adalah ranah keinformatikaan yang melibatkan pembuatan sebuah situs web untuk Internet (World Wide Web) atau Intranet (Jaringan pribadi). Pengembangan Web memiliki definisi yang luas. Satu webpage sederhana sudah termasuk pengembangan web. Dan dari satu webpage bisa dikembangkan menjadi berbagai webpage dan pada akhirnya menjadi sebuah situs web. Situs-situs web yang bisa dibuat berupa aplikasi web, bisnis *e-commerce*, layanan jaringan sosial.

Pengembangan Web memiliki 3 komponen yang perlu diperhatikan untuk membuatnya. Komponen tersebut terdiri dari Frontend, Backend, dan Database. Ketiga komponen ini mempunyai fungsinya masing-masing.

Pengembangan Frontend adalah pengembangan elemen dari website yang bersifat visual dan interaktif. Biasanya front-end yang ideal adalah front-end yang bisa memberikan pengguna daya tarik dan kenyamanan yang membuat pengguna lebih tertarik untuk menggunakan situs web yang sedang digunakan tersebut. Front-end biasanya menggunakan 3 bahasa yaitu HTML, CSS, dan JavaScript. HTML berurusan dengan struktur dari laman web, CSS berurusan dengan tampak dan desain dari laman web, dan JavaScript berurusan dengan interaktivitas dari laman web dan bagaimana cara laman web tersebut bekerja. Selain bahasa pemrograman, Front-End memiliki

sejumlah *framework* dan *library* yang bisa digunakan untuk membantu pengembang untuk membuat laman web. Contohnya adalah Angular, React, Bootstrap, jQuery, SASS, Flutter, Svelte, dll.

Pengembangan Backend adalah pengembangan website pada bagian server. Backend menyimpan dan menyusun data dan memastikan semua bagian client di website bekerja dengan baik. Bagian Backend tidak bisa dilihat oleh pengguna dan juga tidak bisa digunakan. Namun, fitur-fitur dan hasil dari pemrosesan oleh Backend bisa diakses oleh pengguna dengan menggunakan Frontend yang ada di laman web tersebut. Selain membuat program di belakang layar, pembuatan API, library, dll juga merupakan pengembangan Backend. Backend juga mempunyai bahasa-bahasa pemrograman yang bisa digunakan untuk membuatnya. Contohnya adalah PHP, C++, Java, Python, Node.js. Tidak hanya Frontend, Backend juga mempunyai *framework* yang bisa membantu pengembang untuk membuat sebuah laman web. Contohnya adalah Express, Django, Ruby on Rails, Laravel, Spring, dll.

Pengembangan Database adalah pengembangan website pada bagian penyimpanan. Database menyimpan informasi tentang pengguna, gambar, dan data-data lainnya. Database itu adalah komponen yang sangat penting dari situs web apapun karena menyediakan lokasi pusat untuk menyimpan informasi-informasi penting yang dibutuhkan untuk diproses. Tidak hanya itu, Database mempunyai kemampuan untuk menyimpan tipe-tipe data yang lebih kompleks dengan mudah. Database juga mempunyai bahasa-bahasa pemrograman yang bisa digunakan. Contohnya adalah The Oracle, MySQL, MS SQL Server, PostgreSQL, MongoDB, dll.

BAB III
Analisis Pemecahan Masalah

1. Langkah-Langkah Pemecahan Masalah

Dengan menggunakan aplikasi berbasis web, masalah dapat diselesaikan dengan berbagai tahap. Tahapan-tahapan yang perlu diselesaikan secara garis besar adalah cara mengoperasikan aplikasi web untuk membuka laman unggahan, bagaimana gambar tersebut diproses, dll. Berikut adalah tahapannya.

1. Mengakses laman web

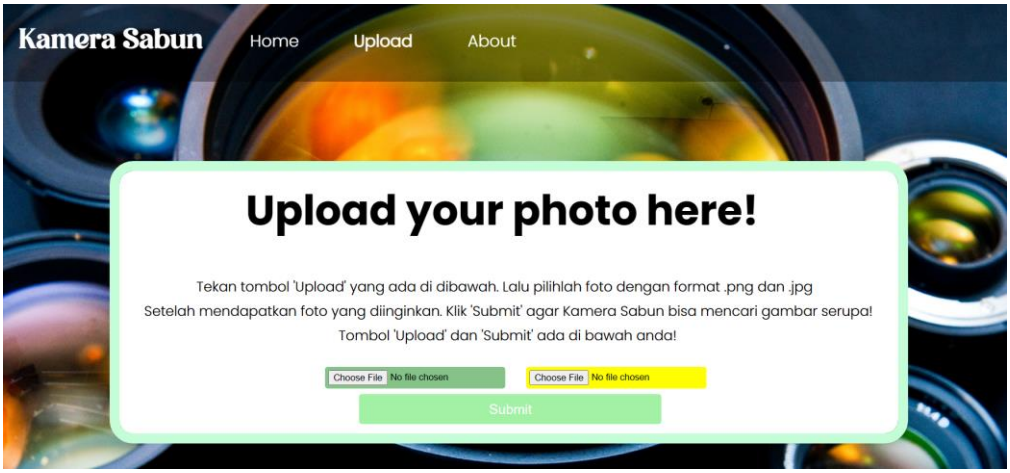
   Biasanya, untuk membuka laman web, hanya perlu membuka website di *software* internet yang diinginkan. Namun, jika laman web belum dideploy, pembukaan aplikasi web dapat dilakukan dengan menggunakan terminal atau dibuka secara langsung. Ada banyak cara untuk membuka laman web tersebut dan itu semua tergantung kepada framework yang digunakan. Misalnya, jika menggunakan vanilla (hanya HTML5, CSS, dan JS saja), maka untuk membuka laman web hanya perlu membuka file .html-nya saja. Sebaliknya, jika menggunakan framework, maka perlu memasukkan *command* tertentu untuk mengeksekusi program-nya. Contohnya adalah React.js yang menggunakan "npm start" untuk menjalankan program tersebut. Setelah program tersebut dijalankan, maka akan tampak gambar seperti ini.

Gambar 3.1 - Contoh halaman home dari website

Beginilah tampak depan dari sebuah website. Di dalam gambar tersebut terdapat 4 tombol yaitu, "Home", "Upload", dan "About". Untuk mengakses laman unggahan, tekan tombol "Upload".

2. Operasi di laman unggahan

   Setelah menekan tombol "Upload" tersebut, akan tampak halaman web untuk mengunggah gambar yang ingin dicari kesamaannya. Contoh dari halaman web tersebut adalah sebagai berikut.



Gambar 3.2 - Contoh laman unggah dari website

Halaman unggah dari website CBIR mempunyai fitur sebagai berikut. Pertama, fitur untuk mengupload dataset. Dataset terdiri dari sebuah folder atau kumpulan file yang akan menjadi patokan bagi sebuah foto yang akan diproses. Selanjutnya, adalah opsi untuk memilih gambar. Gambar yang disubmit adalah berupa 1 file saja. Lalu terakhir ada opsi untuk memilih "Color" dan "Texture". Untuk ini, kita diberi opsi untuk mengoperasikan gambar yang kita submit berdasarkan warna atau tekstur. Setelah semua sudah dipilih, tekan tombol "Submit" untuk memulai proses pencarian gambar.

3. Proses CBIR dari gambar

Ada 2 parameter yang bisa digunakan untuk mengoperasikan CBIR dari gambar yang sudah diinput. Pertama adalah parameter warna dan yang kedua adalah parameter tekstur. Berikut penjelasan dari kedua parameter.

A. CBIR berdasarkan parameter warna

Kasus ini diproses jika dalam laman unggah pengguna memilih opsi "Color". Berikut adalah tahapan singkat dari CBIR histogram warna global.

1. Konversi gambar dari RGB ke HSV
2. Gunakan formula HSV untuk memproses gambar yang masih mengandung RGB
3. Hitung setiap jumlah dari fitur
4. Lalu hitung kesamaan dari jarak Euclidean

Tidak hanya dengan histogram warna global, CBIR dengan parameter warna juga bisa menggunakan cara histogram warna blok. Beginilah cara-cara yang diperlukan untuk memproses gambarnya.

1. Gambar yang diproses dipisah menjadi gambar n x n
2. Setiap blok dilakukan perhitungan konversi ruang warna dan kuantisasi warna.
3. Hitung normalisasi warna untuk setiap blok
4. Disarankan koefisien beban didistribusikan di setiap blok

B. CBIR berdasarkan parameter tekstur

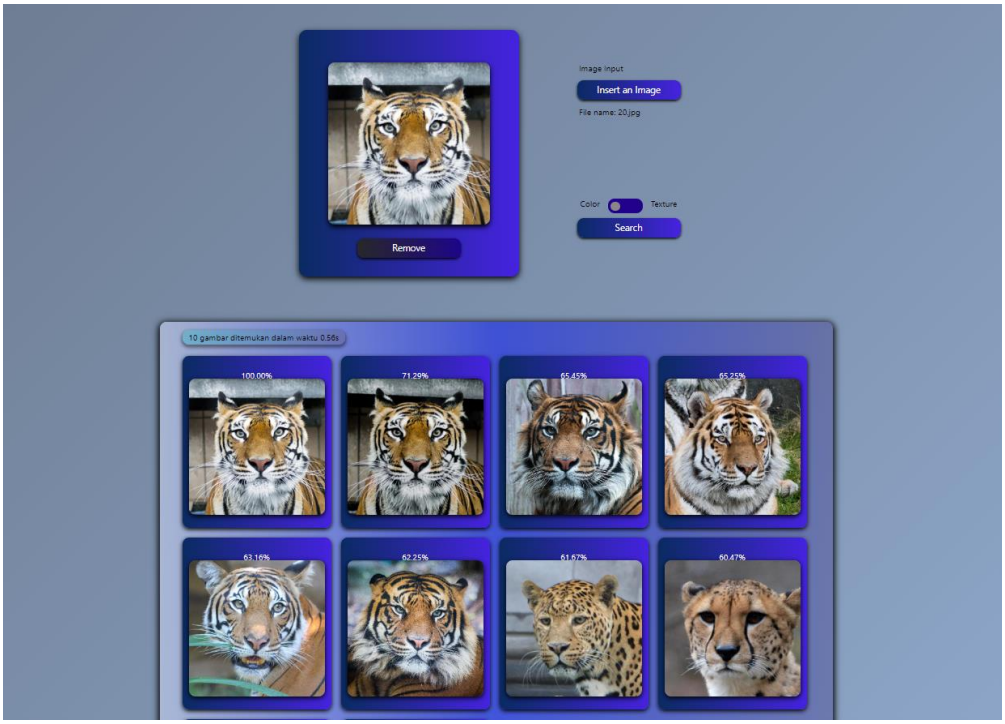
Kasus ini diproses jika dalam laman unggah pengguna memilih opsi “Texture”. Berikut adalah tahapan-tahapan yang akan dilakukan gambar untuk mencari kesamaan berdasarkan tekstur di dataset.

1. Konversi warna gambar menjadi *grayscale*
2. Kuantifikasi nilai *grayscale*
3. Gunakan persamaan yang diperlukan untuk mendapatkan komponen *Contrast*, *Homogeneity*, dan *Entropy*. Lalu carilah *co-occurrence matrix* berdasarkan komponen yang didapat.
4. Hitung kemiripan dari kedua gambar dengan menggunakan Teorema *Cosine Similarity*.

Dari dataset yang diunggah, foto-foto yang ada disitu satu persatu diproses dan dibandingkan dengan foto yang pengguna unggah untuk mencari kesamaannya.

4. Hasil dari proses CBIR gambar

Setelah foto sudah dibandingkan oleh semua foto yang ada di dalam dataset, program akan menerima dan menunjukkan semua gambar yang memiliki kemiripan >60%. Dari program, akan ditunjukkan waktu yang dibutuhkan untuk memproses semua gambar tersebut dan persentase kemiripan dari gambar-gambar yang ditunjukkan. Beginilah contoh dari hasil proses CBIR gambar.



Gambar 3.3 - Contoh hasil dari CBIR

Laman web biasanya memiliki pagination untuk menunjukkan gambar dengan jumlah yang banyak. Pengguna bisa mengoperasikan pagination

tersebut untuk mencari semua gambar yang memiliki kemiripan 60%. Dengan ini, gambar-gambar yang ditunjukan tidak muncul sampai ke halaman yang paling bawah. Setelah melakukan pencarian ini, pengguna boleh melakukan pencarian lagi dengan mengunggah dataset dan gambar yang lain.

2. Proses Pemetaan Masalah Menjadi Elemen-Elemen Pada Aljabar Geometri

CBIR dibagi menjadi 2 yaitu parameter warna dan tekstur. Kedua parameter ini bisa diproses menjadi elemen-elemen aljabar geometri. Pertama, akan dijelaskan tentang CBIR dengan parameter warna.

A. CBIR dengan parameter warna

Ada dua cara untuk mencari kemiripan sebuah gambar dengan menggunakan CBIR dengan parameter warna. Keduanya melibatkan penggunaan konversi RGB ke HSV. Perbedaannya adalah satu menggunakan Euclidean Distance dan satunya lagi menggunakan Teorema *Cosine Similarity*. Berikut penjelasannya.

1. Euclidean Distance
   a. Konversi gambar dari ruang RGB ke ruang HSV
   b. Gunakan formula berikut ini untuk melakukan kuantifikasi pada konversi RGB ke HSV

$$H = \begin{cases} 0 & h \in [316, 360] \\ 1 & h \in [1, 25] \\ 2 & h \in [26, 40] \\ 3 & h \in [41, 120] \\ 4 & h \in [121, 190] \\ 5 & h \in [191, 270] \\ 6 & h \in [271, 295] \\ 7 & h \in [295, 315] \end{cases}$$

$$S = \begin{cases} 0 & s \in [0, 0.2) \\ 1 & s \in [0.2, 0.7) \\ 2 & s \in [0.7, 1] \end{cases}$$

$$V = \begin{cases} 0 & v \in [0, 0.2) \\ 1 & v \in [0.2, 0.7) \\ 2 & v \in [0.7, 1]. \end{cases}$$

   c. Hitung setiap nilai fitur
   d. Hitung kemiripan dengan menggunakan rumus Euclidean Distance yaitu

$$D = \sum_{i=1}^{n} (A_i - B_i)^2.$$

Pada bagian inilah aljabar geometri diaplikasikan. Gambar yang digunakkan untuk melakukan searching dijadikan sebuah vektor yang direpresentasikan dengan A sedangkan Gambar-gambar yang diperiksa kemiripannya satu satu direpresentasikan oleh B.

Setiap vektor A dan B jaraknya dihitung dengan menggunakan Euclidean Distance Formula.

2. Teorema *Cosine Similarity*
    a. *Range* dari RGB diubah dari nilai 0 sampai 255 menjadi 0 sampai 1. Oleh karena itu, semua nilai RGB yang ada di dalam gambar dibagi 255 untuk mendapatkan *range* dari 0 sampai 1 tersebut.
    b. Carilah Cmax, Cmin, dan Δ dengan menggunakan rumus sebagai berikut

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

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

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

    c. Gunakan Cmax, Cmin, dan Δ untuk mendapatkan nilai HSV serta menggunakan rumus sebagai berikut

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

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

$$V = Cmaks$$

    d. Hitung kemiripan dengan menggunakan *cosine similarity*. Berikut adalah rumusnya.

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

Pada bagian inilah aljabar geometri diaplikasikan. Gambar yang digunakkan untuk melakukan searching dijadikan sebuah vektor yang direpresentasikan dengan A sedangkan Gambar-gambar

yang diperiksa kemiripannya satu satu direpresentasikan oleh B. Setiap vektor A dan B jaraknya dihitung dengan menggunakan *Cosine Similarity*.

B. CBIR dengan parameter tekstur

Ada dua bagian penting yang keduanya perlu dilakukan untuk mendapatkan hasil berdasarkan parameter tekstur. Pertama adalah menghitung *co-occurrence matrix* lalu melakukan ekstraksi pada fitur dari tekstur. Disini akan dijelaskan bagaimana kedua bagian tersebut bekerja dan bagaimana elemen-elemen dalam gambar bisa diproses ke dalam aljabar geometri.

1. *Co-occurrence matrix*

Misalkan gambar yang dimasukkan memiliki ukuran m x n pixel dengan parameter offset $(\Delta x, \Delta y)$, maka 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.

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

Dari rumus ini, *co-occurrence matrix* akan didapat. Dan dari matriks yang didapat tersebut, akan dibuat *symmetric matrix* yang merupakan jumlah dari *co-occurrence matrix* dengan hasil dari transpose-nya. Lalu cari *matrix normalization* dengan persamaan.

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

2. Ekstraksi fitur dari tekstur

Ada beberapa tahapan yang perlu dilakukan untuk melakukan ekstraksi. Berikut adalah tahapannya

a. Konversi gambar menjadi grayscale

Sebuah gambar yang berwarna bisa dikonversi menjadi gambar grayscale. Inilah rumus yang bisa digunakan untuk mendapatkan hasil grayscale tersebut.

Y = 0.29 x R + 0.587 x G + 0.114 x B

b. Kuantifikasi grayscale

Skala grayscale gambar adalah 256. Oleh karena itu, *co-occurrence matrix*-nya adalah 256 x 256. Menurut fitur visual manusia, kesamaan sebagian besar gambar dibedakan oleh fitur tekstur yang relatif kasar. Skala grayscale dari gambar awal akan dikompresi untuk mengurangi perhitungan sebelum *co-*

occurrence terbentuk. 16 tingkat kompresi dipilih untuk meningkatkan kecepatan ekstraksi fitur dari tekstur.

c. Menghitung *capacity*, *homogeneity*, dan *entropy*

Gunakkan rumus dibawah untuk mencari 3 komponen ekstraksi tekstur.

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 :

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

P merepresentasikan *co-occurrence matrix*.

Berdasarkan konsep aljabar geometri, dibuatlah sebuah vektor yang akan digunakan dalam proses pengukuran tingkat kemiripan.

d. Ukur kemiripan

Ada dua opsi yang bisa digunakan untuk mencari kemiripan dari kedua gambar. Opsi pertama adalah menggunakan Euclidean Distance dan yang kedua adalah menggunakan Teorema *Cosine Similarity*

1. Euclidean Distance

   Untuk menggunakan Euclidean Distance, gunakan normalisasi internal terlebih dahulu dengan menggunakan Gaussian normalization.

   $$h^{i,j'} = \frac{h^{i,j} - m_j}{\sigma_j}$$

   m merepresentasikan rata-rata, σ melambangkan standar deviasi, dan h melambangkan vektor-vektor yang lama.

Setelah semuanya didapat, baru gunakan Euclidean Distance formula.

$$D = \sum_{i=1}^{n} (A_i - B_i)^2.$$

2. Teorema *Cosine Similarity*
   Pada bagian ini, gunakan rumus sebagai berikut untuk mencari kemiripan.

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

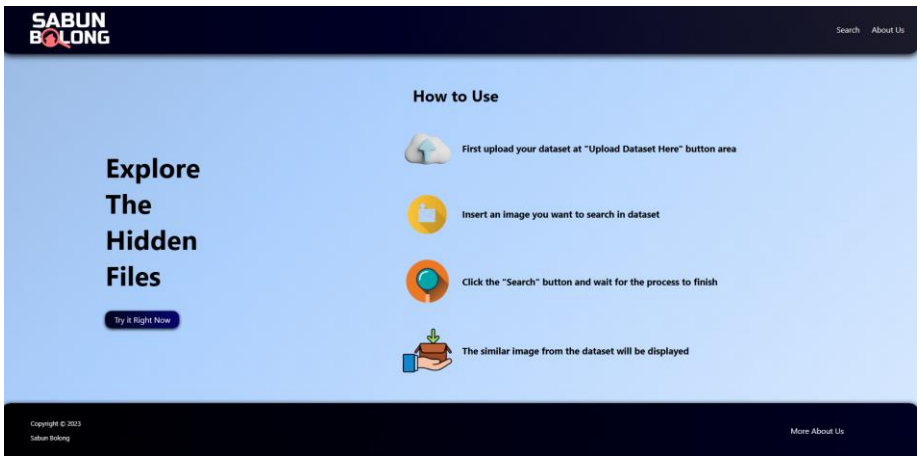
   A dan B adalah 2 vektor dari dua gambar. Semakin besar hasil *Cosine Similarity* kedua vektor maka tingkat kemiripannya semakin tinggi.

3. Contoh Ilustrasi Kasus dan Penyelesaiannya

Diberikan sebuah gambar yang akan diunggah dan sebuah dataset yang menjadi patokan untuk melakukan image search. Beginilah tampak dari pemecahan masalahnya berdasarkan 2 parameter yaitu warna dan tekstur.

A. CBIR berdasarkan warna

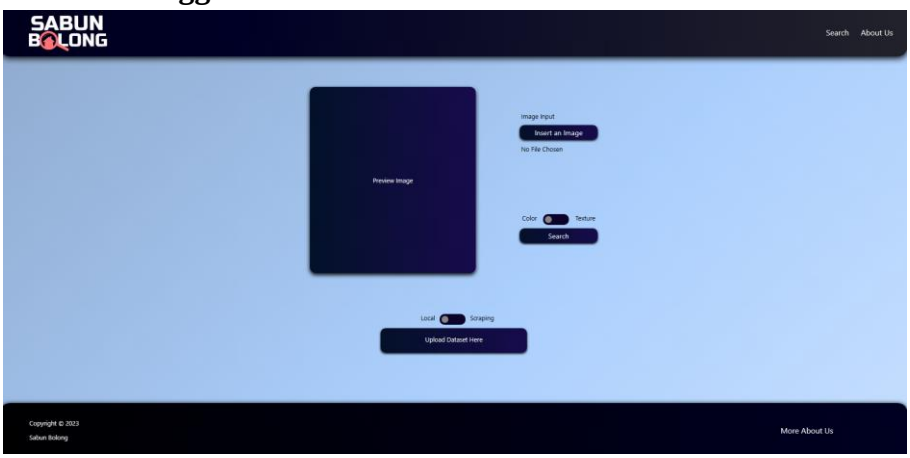
1. Halaman Awal



Gambar 3.5 - Halaman awal dari website

Dari sini, tekan tombol “Upload” untuk pergi ke halaman “Upload.

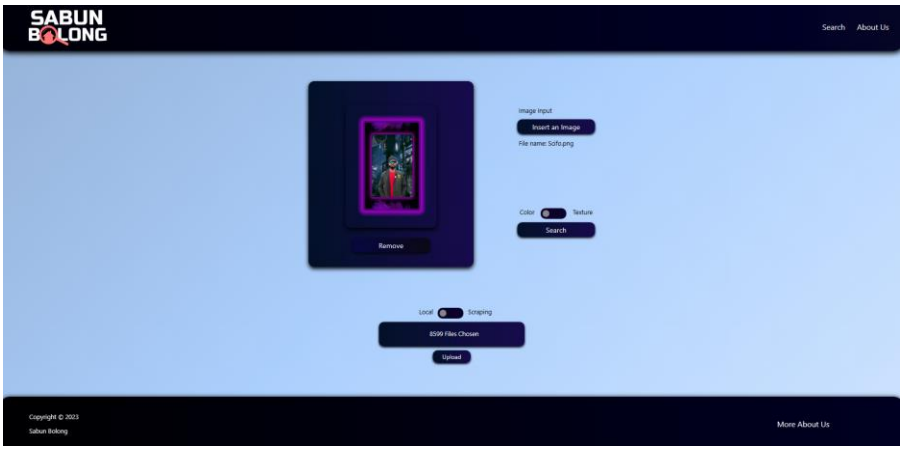
2. **Halaman Unggah**



Gambar 3.6 - Halaman untuk mengunggah gambar

Pada halaman ini, tombol "Upload Dataset Here" ditekan untuk mengunggah sebuah dataset.

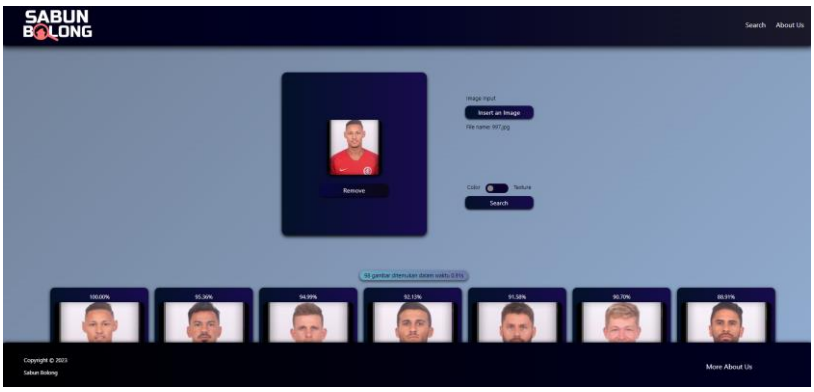
3. **Pengunggahan Dataset**



Gambar 3.7 - Dataset sedang diproses

Setelah ini, gambar akan diunggah lalu diproses. Perlu diperhatikan bahwa CBIR di-*toggle* ke *mode* "Color". Ini artinya gambar akan diproses secara "Color".

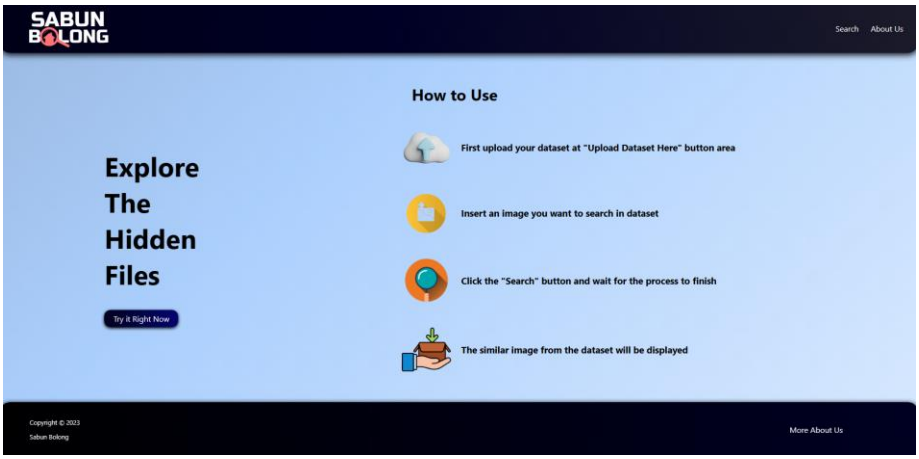
4. **Hasil**



Gambar 3.8 - Hasil CBIR dengan parameter warna

B. CBIR berdasarkan tekstur

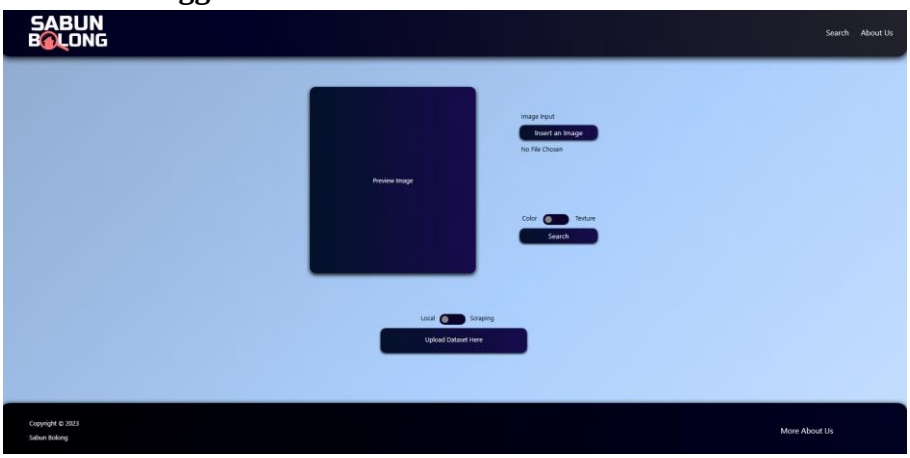
1. **Halaman Awal**



Gambar 3.9 - Halaman awal dari website

Dari sini, tekan tombol “Upload” untuk pergi ke halaman “Upload.

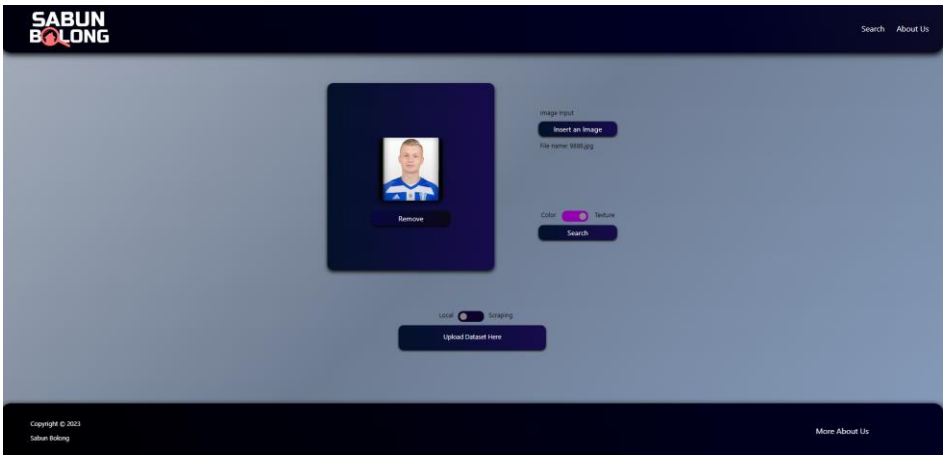
2. **Halaman Unggah**



Gambar 3.10 - Halaman untuk mengunggah gambar

Pada halaman ini, tombol "Upload Dataset Here" ditekan untuk mengunggah sebuah dataset.

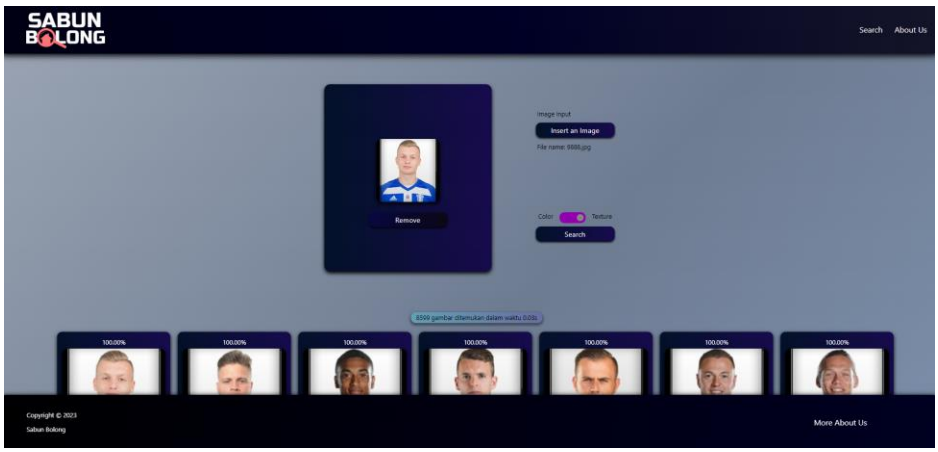
3. **Pengunggahan Dataset**



Gambar 3.11 - Dataset sedang diproses

Setelah ini, gambar akan diunggah lalu diproses. Perlu diperhatikan bahwa CBIR di-*toggle* ke *mode* "Texture". Ini artinya gambar akan diproses secara "Texture"

4. Hasil



Gambar 3.12 - Hasil CBIR dengan parameter warna

BAB IV
Eksperimen

1. Implementasi Program Utama

Program utama yang kami punya terdiri dari 2 yaitu program untuk melakukan CBIR berdasarkan warna dan CBIR berdasarkan tekstur.

```
// CBIR FOR COLOR
FUNCTION ConvertHSV(paint)
    // Convert a color to HSV values
    // Retrieve RGB values from the color
    R, G, B = GetRGBValuesFromColor(paint)
    // Normalize RGB values
    R = R / 255
    G = G / 255
    B = B / 255
    // Calculate HSV values
    V = Max(R, Max(G, B))
    min = Min(R, Min(G, B))
    delta = V - min
    IF V == 0 THEN
        S = 0
    ELSE
        S = delta / V
    END IF
    IF delta == 0 THEN
        H = 0
    ELSE IF V == R THEN
        H = (G - B) / delta
        IF H < 0 THEN
            H += 6
        END IF
    ELSE IF V == G THEN
        H = (B - R) / delta + 2
    ELSE
        H = (R - G) / delta + 4
    END IF
    H *= 60
    RETURN H, S, V

FUNCTION IndexHSV(paint)
    // Convert a color to an index based on its HSV values
    h, s, v = ConvertHSV(paint)
    // Determine HSV ranges and assign corresponding indices
    // ... (conditions for assigning H, S, V values to
indices)
    RETURN H*9 + S*3 + V

FUNCTION LengDorm(bins)
    // Calculate Euclidean norm (length) of an array
    temp = 0
    FOR i = 0 TO 71
        temp += bins[i] * bins[i]
```

```
    END FOR
    ret = SquareRoot(temp)
    RETURN ret

FUNCTION ReadImage(path)
    // Read an image file from the specified path
    // Open the file
    // Decode the image
    RETURN image

STRUCT Vektor
    ImgName STRING
    Bins ARRAY[16][72] OF INT

FUNCTION CalculateVectorThread(path, resultChan, wg)
    // Read an image from the given path
    img = ReadImage(path)
    IF img is NULL THEN
        Display "Image is not extracted."
        RETURN
    END IF
    // Create Vektor struct
    vektor.ImgName = path
    // Calculate image vector bins based on HSV values
    // ... (loop through image blocks, calculate HSV values,
and update bins)
    resultChan <- vektor

FUNCTION CreateDataExtraction(dirpath)
    // Read images in the directory and extract data vectors
    // ... (loop through image files, spawn
CalculateVectorThread goroutines)
    RETURN data

FUNCTION PreprocessImageColor(dirpath, destFile)
    // Perform preprocessing to extract image color data
    data = CreateDataExtraction(dirpath)
    // Write extracted data to a file in JSON format
    // ... (create file, encode data into JSON, and write to
the file)

FUNCTION IsImage(file)
    // Check if the file is an image file (JPG, JPEG, PNG)
    // Extract file extension and compare
    RETURN IsExtensionImage(file.Extension())

FUNCTION ListImageInDir(dirpath)
    // List image files in a directory
    // Read directory contents
    // ... (loop through files, count and filter image files)

FUNCTION CalculateSingleVector(path)
    // Calculate vector for a single image file
    // ... (similar logic to CalculateVectorThread without
concurrency)
```

```
FUNCTION ExtractColorVector(filePath)
    // Extract color vectors from a JSON file
    // ... (open file, decode JSON data into Vektor structs)

STRUCT tuplePercentage
    Path STRING
    Percentage FLOAT

FUNCTION SearchImageColor(imageSearched, binsFile,
targetFile)
    // Search for similar images based on color
    // ... (similar logic to SearchImageColor in Go with
goroutines and concurrency)
```

```
// CBIR FOR TEXTURE
STRUCT ImgComp
    URL        STRING
    Contrast   FLOAT
    Homogenity FLOAT
    Entropy    FLOAT

STRUCT ImgSim
    URL        STRING
    Similarity FLOAT

STRUCT GlcmMatrix
    occMtrx             [][]FLOAT64
    lowerBound, sizeOcc INT

STRUCT GrayImg
    grayScale           [][]UINT8
    widthImg, heightImg INT
    minGray, rangeGray  UINT8

VAR textureArr []ImgComp

FUNCTION MakeTexture(url STRING) ImgComp
    ImgGray = MakeGray(url)
    glcm = MakeOcc(ImgGray, 1, 0)
    imgComp = MakeImgComp(glcm)
    imgComp.URL = GetBasePath(url)
    RETURN imgComp

FUNCTION ListFiles(root STRING) ([]os.DirEntry, error)
    dirEntries, err = os.ReadDir(root)
    IF err != nil THEN
        RETURN nil, err
    END IF

    files = []
    FOR EACH entry IN dirEntries
        APPEND entry TO files
    END FOR
    RETURN files, nil
```

```
FUNCTION MakeJSONDataset(root STRING, dest STRING)
    files, err = ListFiles(root)
    IF err != nil THEN
        PRINT "error listing files in directory", root, ":",
err
        RETURN
    END IF

    textureArr = []

    var wg sync.WaitGroup
    var mu sync.Mutex
    prosesFile = map[string]bool{}

    FOR EACH entry IN files
        fp = filepath.Join(root, entry.Name())
        mu.Lock()
        IF NOT prosesFile[fp]
            prosesFile[fp] = true
            mu.Unlock()
            wg.Add(1)
            GO
                imgComp = MakeTexture(fp)
                mu.Lock()
                APPEND imgComp TO textureArr
                mu.Unlock()
                wg.Done()
            END GO
        ELSE
            mu.Unlock()
        END IF
    END FOR

    wg.Wait()
    file, err = os.OpenFile(dest,
os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0644)
    IF err != nil THEN
        PRINT "Error opening file:", err
        RETURN
    END IF

    encoder = json.NewEncoder(file)
    err = encoder.Encode(textureArr)
    IF err != nil THEN
        PRINT "Error encoding JSON:", err
        RETURN
    END IF

FUNCTION MakeGray(url STRING) GrayImg
    var imG GrayImg
    inputFile, err = os.Open(url)
    IF err != nil THEN
        PRINT "Error opening image:", err
        RETURN imG
```

```
    END IF
    DEFER inputFile.Close()

    img, _, err = image.Decode(inputFile)
    IF err != nil THEN
        RETURN imG
    END IF

    minGray, maxGray = 255, 0
    size = img.Bounds().Size()
    grayScale = []

    FOR i FROM 0 TO size.Y
        colArr = []
        FOR j FROM 0 TO size.X
            pixelColor = img.At(i, j)
            rgbaColor =
color.RGBAModel.Convert(pixelColor).(color.RGBA)
            colGray = uint8(0.29*float64(rgbaColor.R) +
0.587*float64(rgbaColor.G) + 0.114*float64(rgbaColor.B))
            IF colGray > maxGray THEN
                maxGray = colGray
            END IF
            IF colGray < minGray THEN
                minGray = colGray
            END IF
            APPEND colGray TO colArr
        END FOR
        APPEND colArr TO grayScale
    END FOR

    imG.grayScale = grayScale
    imG.minGray = minGray
    imG.rangeGray = maxGray - minGray + 1
    imG.widthImg = size.X
    imG.heightImg = size.Y
    RETURN imG

FUNCTION MakeOcc(imG GrayImg, xOffset INT, yOffset INT)
GlcmMatrix
    var glcm GlcmMatrix
    glcm.sizeOcc = int(imG.rangeGray)
    glcm.lowerBound = int(imG.minGray)
    glcm.occMtrx = [][]float64{}

    FOR i FROM 0 TO glcm.sizeOcc
        rowArr = []
        FOR j FROM 0 TO int(imG.rangeGray)
            APPEND 0 TO rowArr
        END FOR
        APPEND rowArr TO glcm.occMtrx
    END FOR

    glcmValue = 0
```

```
    FOR i FROM 0 TO imG.heightImg
        FOR j FROM 0 TO imG.widthImg
            grayFirst = imG.grayScale[i][j]
            neighborY = i + yOffset
            neighborX = j + xOffset
            neighborValid = neighborX < imG.widthImg &&
neighborY < imG.heightImg && neighborX >= 0 && neighborY >= 0
            IF neighborValid THEN
                graySecond =
imG.grayScale[neighborY][neighborX]
                glcm.occMtrx[grayFirst-
imG.minGray][graySecond-imG.minGray] += 1
            END IF
        END FOR
    END FOR

    FOR i FROM 0 TO glcm.sizeOcc
        FOR j FROM 0 TO glcm.sizeOcc
            glcmValue += int(glcm.occMtrx[i][j])
        END FOR
    END FOR

    FOR i FROM 0 TO glcm.sizeOcc
        FOR j FROM 0 TO glcm.sizeOcc
            glcm.occMtrx[i][j] /= float64(glcmValue)
        END FOR
    END FOR
    RETURN glcm

FUNCTION MakeImgComp(glcm GlcmMatrix) ImgComp
    var ImgTemp ImgComp
    FOR i FROM 0 TO glcm.sizeOcc
        FOR j FROM 0 TO glcm.sizeOcc
            IF glcm.occMtrx[i][j] > 0 THEN
                ImgTemp.Contrast += glcm.occMtrx[i][j] *
float64((i-j+2*glcm.lowerBound)*(i-j+2*glcm.lowerBound))
                ImgTemp.Homogenity += glcm.occMtrx[i][j] /
(float64(1 + (i-j+2*glcm.lowerBound)*(i-
j+2*glcm.lowerBound)))
                ImgTemp.Entropy -= glcm.occMtrx[i][j] *
math.Log10(glcm.occMtrx[i][j])
            END IF
        END FOR
    END FOR
    RETURN ImgTemp

FUNCTION ImgCompLength(imgComp ImgComp) FLOAT
```

2. Penjelasan Struktur Program

Pada tubes ini, program yang kami buat terdiri dari 2 komponen yaitu front-end dan back-end. Hal ini disebabkan karena program yang kami buat adalah berupa situs web. Oleh karena itu, kami membutuhkan komponen tersebut. Front-end dibuat agar pengguna situs web memiliki pengalaman yang baik saat menelusuri program kami dan

back-end dibuat agar program bisa berjalan dengan lancar tanpa terbatasi oleh bahasa pemrograman dari front-end.

Pada front-end dan back-end yang kami buat, kami menggunakan 1 framework dan 1 bahasa pemrograman. Pada front-end kami menggunakan framework yang bernama React.js. React adalah pustaka JavaScript front-end sumber terbuka dan gratis untuk membangun antarmuka pengguna berdasarkan komponen. React dapat digunakan untuk mengembangkan aplikasi satu halaman, seluler, atau yang direnderi server dengan kerangka kerja seperti Next.js. Karena React hanya mementingkan antarmuka pengguna dan rendering komponen ke DOM, aplikasi React sering kali mengandalkan pustaka untuk perutean dan fungsionalitas sisi klien lainnya.

Pada back-end kami menggunakan bahasa pemrograman Go. Go adalah bahasa pemrograman tingkat tinggi yang dikompilasi dan diketik secara statis yang dirancang di Google oleh Robert Griesemer, Rob Pike, dan Ken Thompson. Secara sintaksis mirip dengan C, tetapi juga memiliki keamanan memori, pengumpulan sampah, pengetikan struktural, dan konkurensi gaya CSP. Alasan kami menggunakan Go adalah karena jalannya program relatif lebih cepat dibandingkan bahasa pemrograman lainnya.

3. Penjelasan Tata Cara Penggunaan Program

Sebelum menggunakan program, jika program dipasang langsung di file local, maka ada beberapa hal yang harus anda lakukan. Berikut tahapan-tahapannya

1. Ketik cd <File path> untuk mengakses direktori dari file program
2. Jangan lupa menginstall Go di dalam komputer anda
3. Ketik npm install pada terminal anda
4. Ketik npm install <dependencies> pada terminal anda. Tag <dependencies> merujuk kepada dependency yang dibutuhkan agar program berjalan. Cek dependency di file go.mod dan package.json.
5. Ketik npm start agar anda bisa dibawa ke local.host:xxxx.
6. Setelah itu gunakanlah situs web tersebut

Jika anda menggunakan program yang sama tetapi sudah dideploy, maka semua dependency dan back-end sudah diatur oleh web.

Setelah memasuki situs web. Maka inilah hal-hal yang perlu anda lakukan untuk menjalankan situs web ini. Berikut adalah cara-caranya.

1. Klik tombol “Search” yang ada di laman web untuk memasuki halaman search
2. Pilihlah opsi dataset yang anda inginkan. Jika ingin memilih lokal, anda bisa mengunggah file dengan menekan tombol “Upload Dataset Here” atau drag folder ke halaman internet. Program akan memproses masukkan tersebut dan menunjukkan jumlah gambar yang akan dijadikan dataset. Jika memilih Scraping, masukkan situs web yang anda inginkan untuk diambil datasetnya dan tekan tombol “Scrape”. Dataset tersebut berisi semua gambar yang ada di halaman tersebut.
3. Jika unggahan sudah selesai, tekan tombol “Insert an Image” untuk mencari gambar yang ingin anda proses dengan dataset-nya. Jangan lupa untuk toggle “Color” atau “Texture” untuk memilih parameter pencarian gambar tersebut.

4. Setelah sudah selesai, tekan tombol “Search” agar program bisa melakukan searching pada gambar yang ingin anda cari kesamaannya.
5. Setelah proses CBIR selesai, gambar-gambar yang memiliki kesamaan >60% akan ditunjukkan.
6. Anda bisa melihat semua gambar tersebut dengan melihat halaman-halaman yang ada. Halaman yang dibuat adalah dalam bentuk pagination.

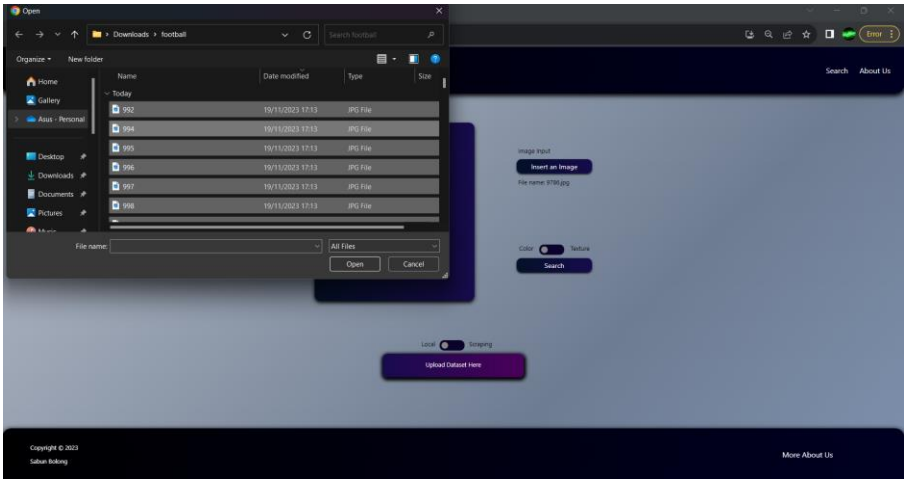
4. Hasil Pengujian

Berikut adalah hasil-hasil eksperimen yang kami lakukan terhadap situs web kami. Kami melakukan eksperimen berdasarkan 2 parameter yaitu parameter “Color” dan parameter “Tekstur”. Masing-masing parameter mempunyai 3 cara untuk melakukan unggahan yaitu unggahan file, unggahan folder, dan web scraping.

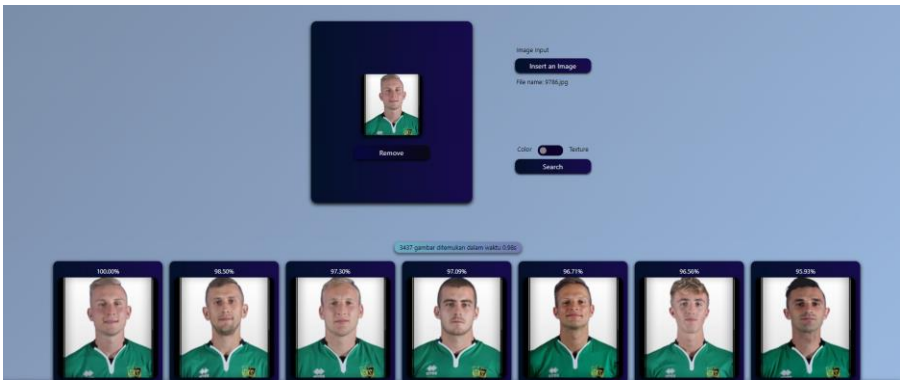
A. Eksperimen Pertama (Color)

1. Local Multiple Files

   Dataset : football (8599 files)



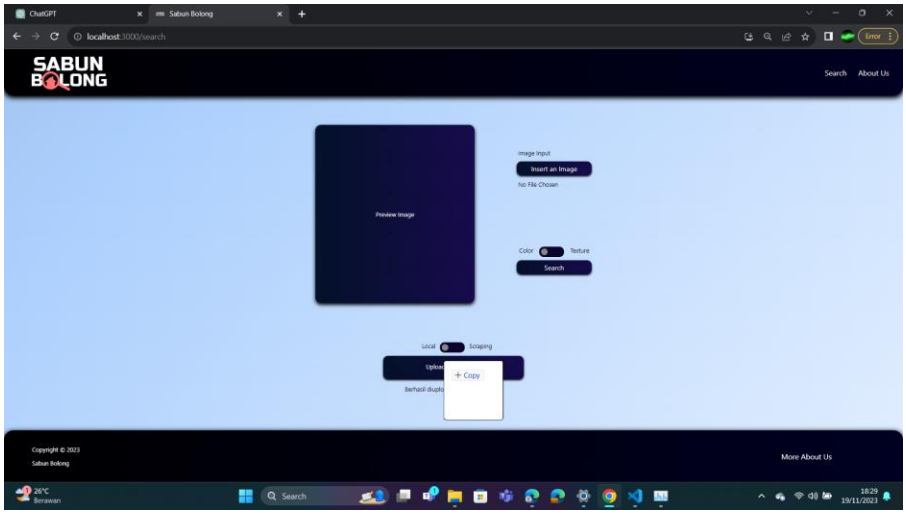
Gambar 4.1 - Memilih File dengan jumlah banyak



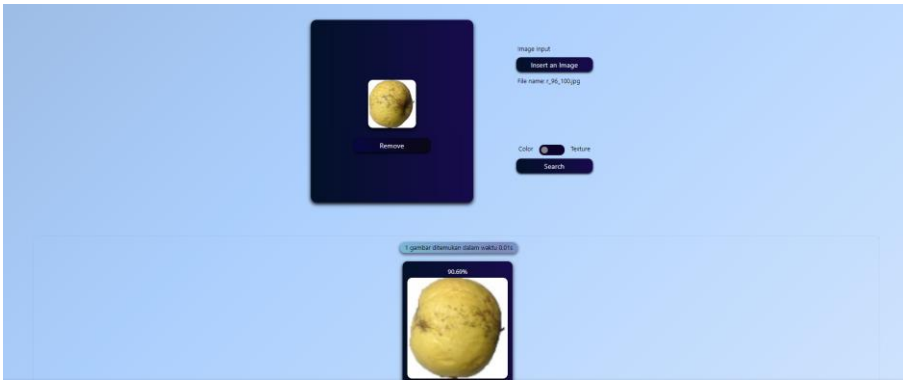
Gambar 4.2 - Hasil Eksperimen

2. Local Folder

   Dataset : Apple Golden (160 Files)

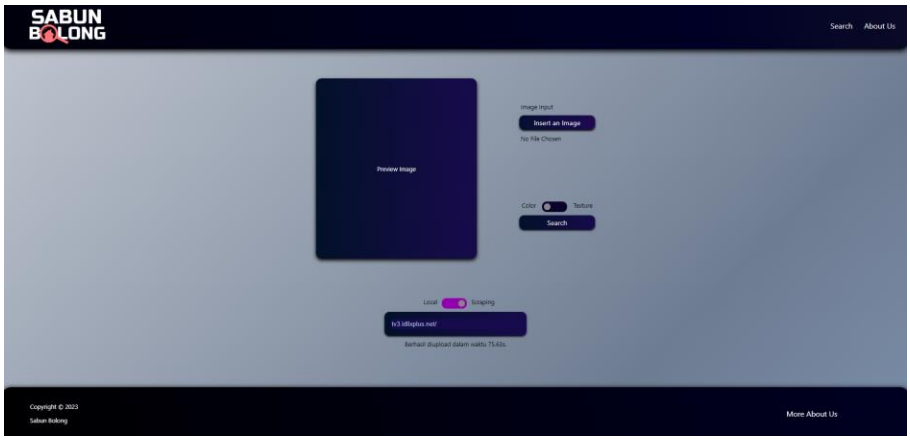


Gambar 4.3 - Unggah Dataset Folder

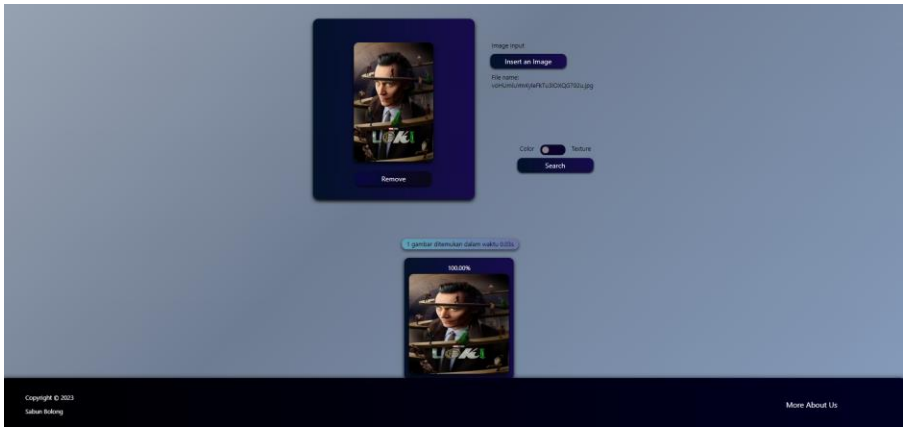


Gambar 4.4 - Hasil Eksperimen

3. Web Scrapping



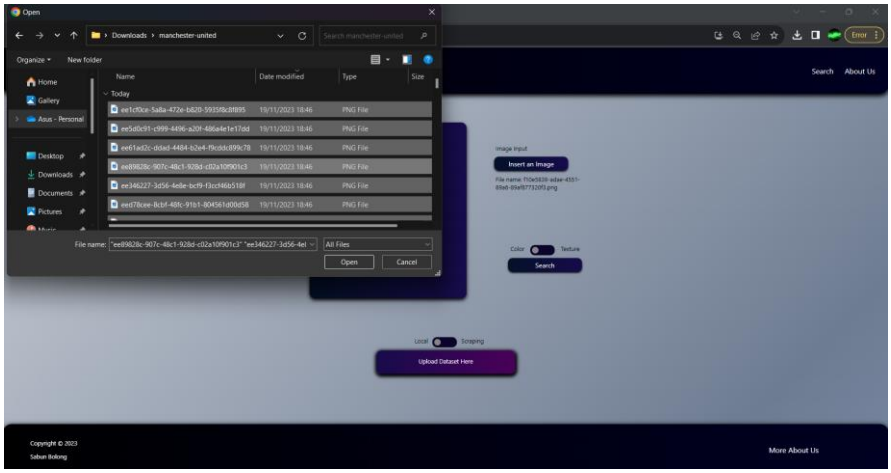
Gambar 4.5 - Request Dari Website Luar



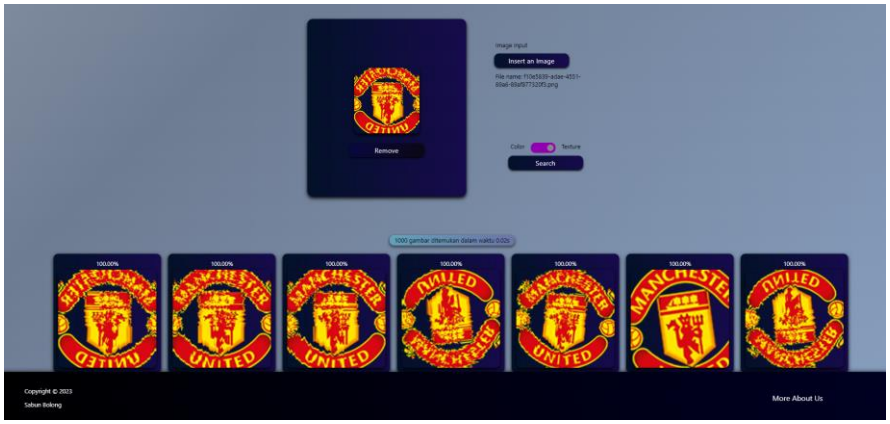
Gambar 4.6 - Hasil Eksperimen

B. Eksperimen Kedua (Texture)

1. Local Multiple Files

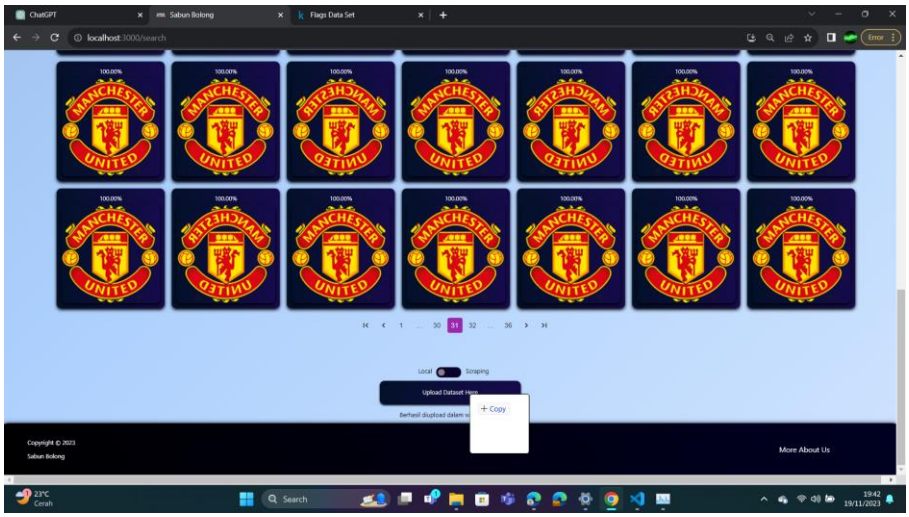


Gambar 4.7 - Unggah File Banyak

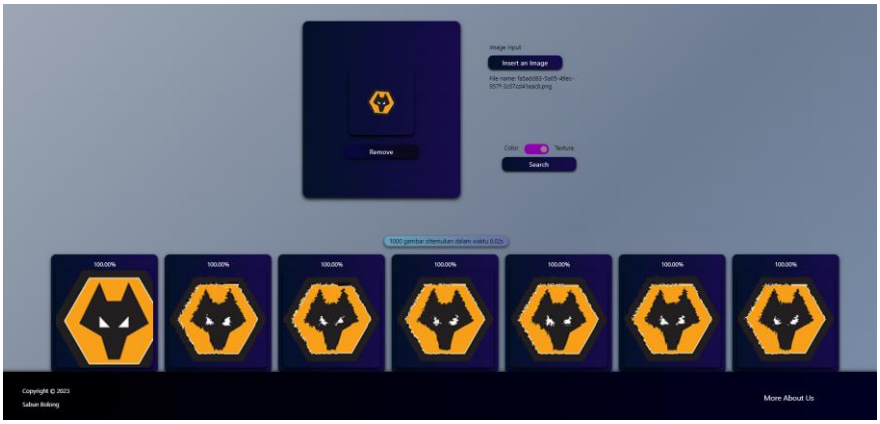


Gambar 4.8 - Hasil Eksperimen

2. Local Folder

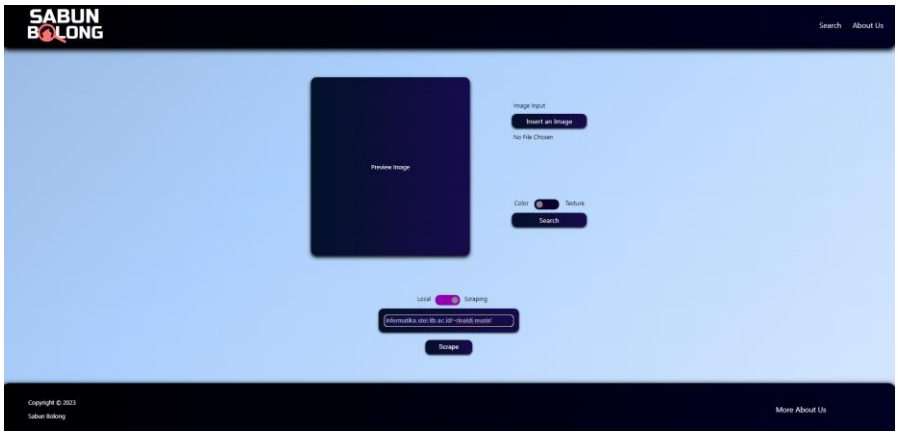


Gambar 4.9 - Unggah Folder

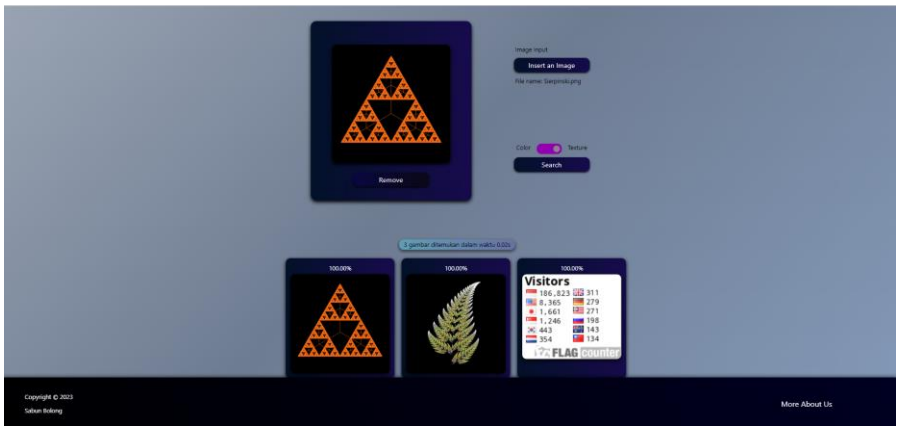


Gambar 4.10 - Hasil Eksperimen

3. Web Scraping



Gambar 4.11 - Web Scraping



Gambar 4.12 - Hasil Eksperimen

5. Analisis Desain Solusi

Berdasarkan hasil CBIR yang kami dapatkan, bisa kami simpulkan bahwa *Content Based Image Retrieval* berdasarkan parameter warna terbukti lebih baik daripada parameter tekstur. Hal ini disebabkan karena salah satu komponen vektor di parameter tekstur jauh lebih besar dibandingkan vektor lainnya. Oleh karena itu, nilai komponen vektor yang lainnya tidak memiliki pengaruh signifikan terhadap "arah" vektor tekstur. Hal ini mengakibatkan ketika kami membandingkan dua buah vektor tekstur, kedua vektor akan menunjuk pada arah yang sama. Berbeda dengan parameter warna, parameter warna memiliki keuntungan berupa hasil perbandingan lewat *Cosine Similarity* lebih baik dibandingkan parameter tekstur. Namun, masih ada kekurangan karena ada kemungkinan gambar yang benar-benar berbeda dianggap sama. Hal ini terjadi karena masih ada kemungkinan komposisi warna yang sama. Oleh karena itu, terbukti bahwa CBIR parameter warna lebih baik daripada parameter tekstur.

BAB V
Penutup

1. Kesimpulan

Berdasarkan implementasi dari program yang telah dirancang, dapat disimpulkan beberapa poin berikut.

1.1. Program berbasis web terdiri dari front-end, back-end, dan database. Front-end memfokuskan tampak dari sebuah situs web dan memberikan pengguna pengalaman yang terbaik ketika menggunakan situs tersebut. Back-end memfokuskan bagaimana sebuah program bekerja.

1.2. Program berbasis web dapat menerima gambar untuk diproses kemiripannya dengan menggunakan *Content Based Image Retrieval* (CBIR). CBIR bisa digunakan dengan parameter warna dan tekstur.

1.3. Program berbasis web ini terdiri dari Home, Upload, dan About. Home berisi pengenalan akan program yang akan dijalankan, Upload berisi tempat untuk mengunggah gambar dan dataset untuk diproses kemiripannya. About berisi informasi tentang pembuat dari situs web.

1.4. Detail dan teknis lebih lanjut mengenai implementasi dari masing-masing fitur dapat dilihat pada bab III dan bab IV.

2. Saran dan Komentar

Ada beberapa saran dan komentar yang bisa kami dapatkan setelah merancang program ini, yakni sebagai berikut.

2.1. Program seharusnya menggunakan metode lain untuk mengimplementasikan kemiripan gambar berdasarkan tekstur karena CBIR kurang efektif dalam memproses kemiripan tekstur gambar.

2.2. Tampak depan website bisa dibuat lebih bagus lagi.

3. Refleksi

Tubes ini adalah media yang baik untuk memperkenalkan kami dengan Web Development. Namun, kami memerlukan waktu yang lebih untuk melakukan eksplorasi akan Web Development baik dari segi front-end maupun back-end. Tubes ini adalah salah satu tubes yang sangat abstrak dibandingkan tubes lainnya karena memerlukan eksplorasi yang lebih banyak dibandingkan tubes-tubes lainnya. Kami harap dengan tubes ini kami bisa meningkatkan kemampuan kami terkait Web Development dan bisa membuat situs web yang lebih baik lagi untuk proyek selanjutnya.

4. Ruang Perbaikan atau Pengembangan

4.1. Perlu *time management* dan *priority management* yang lebih efisien karena untuk mengerjakan tubes ini kami masih harus mengorbankan waktu tidur dan kuliah kami sehingga akademik kami terganggu

4.2. Perlu pengetahuan yang lebih akan Web Development karena sebagian besar waktu yang kami alokasikan untuk mengerjakan tubes ini adalah melakukan eksplorasi agar program yang dibuat tidak salah

DAFTAR REFERENSI

https://www.sciencedirect.com/science/article/pii/S0895717710005352 (Diakses pada 11 November 2023)

https://yunusmuhammad007.medium.com/feature-extraction-gray-level-co-occurrence-matrix-glcm-10c45b6d46a1 (Diakses pada 12 November 2023)

TAUTAN

https://github.com/chankiel/Algeo02-22029 (Tautan Repository Tubes)

https://youtu.be/qR7p5gEKn-E (Tautan Video)