LAPORAN TUGAS BESAR 3
IF2211 STRATEGI ALGORITMA
PENERAPAN *STRING MATCHING* DAN *REGULAR EXPRESSION* DALAM PEMBUATAN CHATGPT SEDERHANA

Kelompok GatauNamanya :

Michael Leon Putra Widhi	(13521108)
Johanes Lee	(13521148)
Nathan Tenka	(13521172)

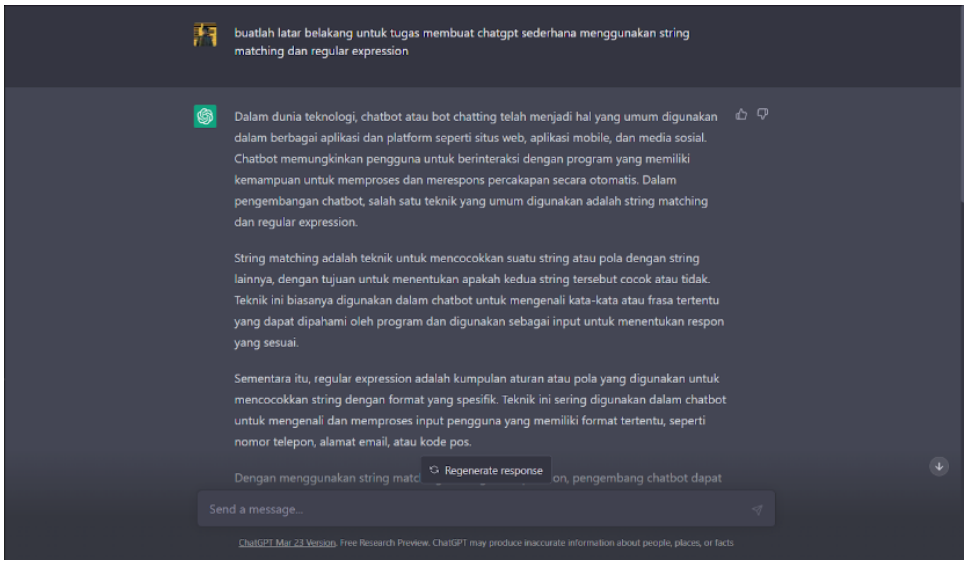
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2023

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Dalam dunia teknologi, *chatbot* telah menjadi hal yang umum digunakan dalam berbagai aplikasi dan *platform* seperti situs web, aplikasi *mobile*, dan media sosial. *Chatbot* memungkinkan pengguna untuk berinteraksi dengan program yang memiliki kemampuan untuk memproses dan merespons percakapan secara otomatis. Salah satu contoh *chatbot* yang sedang *booming* saat ini adalah ChatGPT.



Gambar 1.1. Ilustrasi *Chatbot* ChatGPT (*fun fact* latar belakang spek ini dari ChatGPT)
Sumber: https://chat.openai.com/chat

Pembangunan *chatbot* dapat dilakukan dengan menggunakan berbagai pendekatan dari bidang *Question Answering* (QA). Pendekatan QA yang paling sederhana adalah menyimpan sejumlah pasangan pertanyaan dan jawaban, menentukan pertanyaan yang paling mirip dengan pertanyaan yang diberikan pengguna, dan memberikan jawabannya kepada pengguna. Untuk mencocokkan input pengguna dengan pertanyaan yang disimpan pada *database*, kalian bisa menggunakan *string matching*.

String matching adalah teknik untuk mencocokkan suatu string atau pola dengan string lainnya, dengan tujuan untuk menentukan apakah kedua string tersebut cocok atau tidak. Teknik ini biasanya digunakan dalam *chatbot* untuk mengenali kata-kata atau frasa tertentu yang dapat

dipahami oleh program dan digunakan sebagai input untuk menentukan respon yang sesuai. Sementara itu, *regular expression* adalah kumpulan aturan atau pola yang digunakan untuk pencocokan *string* dengan format yang spesifik. Teknik ini sering digunakan dalam *chatbot* untuk mengenali dan memproses input pengguna yang memiliki format tertentu, seperti nomor telepon, alamat email, atau kode pos.

Dalam tugas besar ini, anda diminta untuk membangun sebuah aplikasi ChatGPT sederhana dengan mengaplikasikan pendekatan QA yang paling sederhana tersebut. Pencarian pertanyaan yang paling mirip dengan pertanyaan yang diberikan pengguna dilakukan dengan algoritma pencocokan *strin*g Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Regex digunakan untuk menentukan format dari pertanyaan (akan dijelaskan lebih lanjut pada bagian fitur aplikasi). Jika tidak ada satupun pertanyaan pada *database* yang *exact match* dengan pertanyaan pengguna melalui algoritma KMP ataupun BM, maka gunakan pertanyaan termirip dengan kesamaan setidaknya 90% Apabila tidak ada pertanyaan yang kemiripannya di atas 90%, maka *chatbot* akan memberikan maksimum 3 pilihan pertanyaan yang paling mirip untuk dipilih oleh pengguna.

Perhitungan tingkat kemiripan dibebaskan kepada anda asalkan dijelaskan di laporan, namun disarankan menggunakan salah satu dari algoritma *Hamming Distance*, *Levenshtein Distance*, ataupun *Longest Common Subsequence*.

Fitur-Fitur Aplikasi:

ChatGPT sederhana yang anda membuat wajib dapat melakukan beberapa fitur / klasifikasi *query* seperti berikut:

1. **Fitur pertanyaan teks (didapat dari *database*)**

   Mencocokkan pertanyaan dari *input* pengguna ke pertanyaan di *database* menggunakan algoritma KMP atau BM.

2. **Fitur kalkulator**

   Pengguna memasukkan *input query* berupa persamaan matematika. Contohnya adalah 2*5 atau 5+9*(2+4). Operasi cukup tambah, kurang, kali, bagi, pangkat, kurung.

3. **Fitur tanggal**

Pengguna memasukkan *input* berupa tanggal, lalu *chatbot* akan merespon dengan hari apa di tanggal tersebut. Contohnya adalah 25/08/2023 maka *chatbot* akan menjawab dengan hari senin.

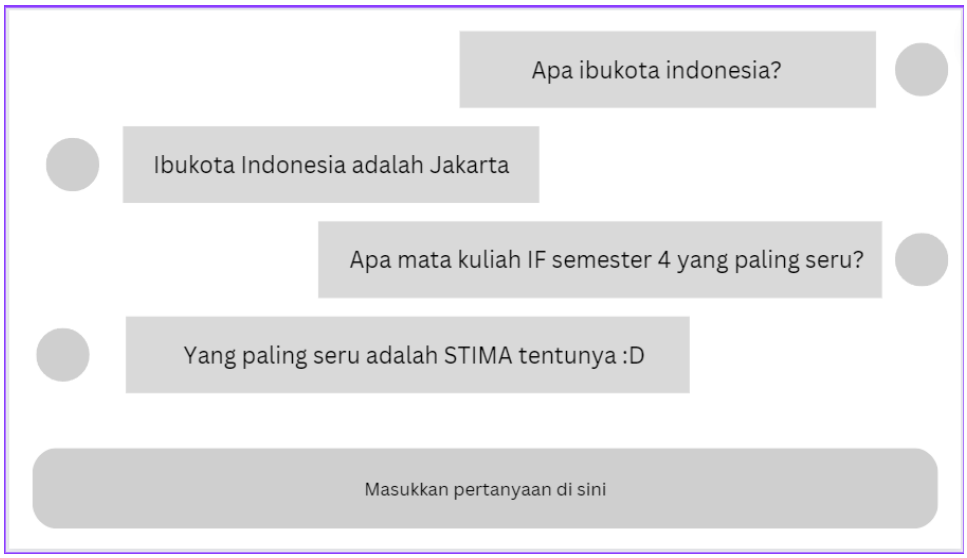
4. **Tambah pertanyaan dan jawaban ke *database***

   Pengguna dapat menambahkan pertanyaan dan jawabannya sendiri ke *database* dengan *query* contoh "Tambahkan pertanyan xxx dengan jawaban yyy". Menggunakan algoritma *string matching* untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbaharui.

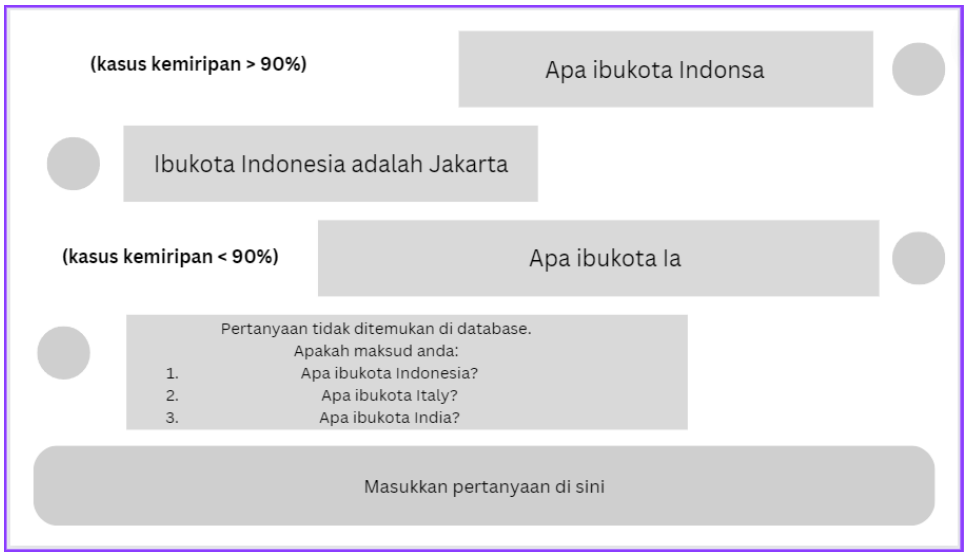
5. **Hapus pertanyaan dari *database***

   Pengguna dapat menghapus sebuah pertanyaan dari *database* dengan *query* contoh "Hapus pertanyaan xxx". Menggunakan *string* algoritma *string matching* untuk mencari pertanyaan xxx tersebut pada *database*.

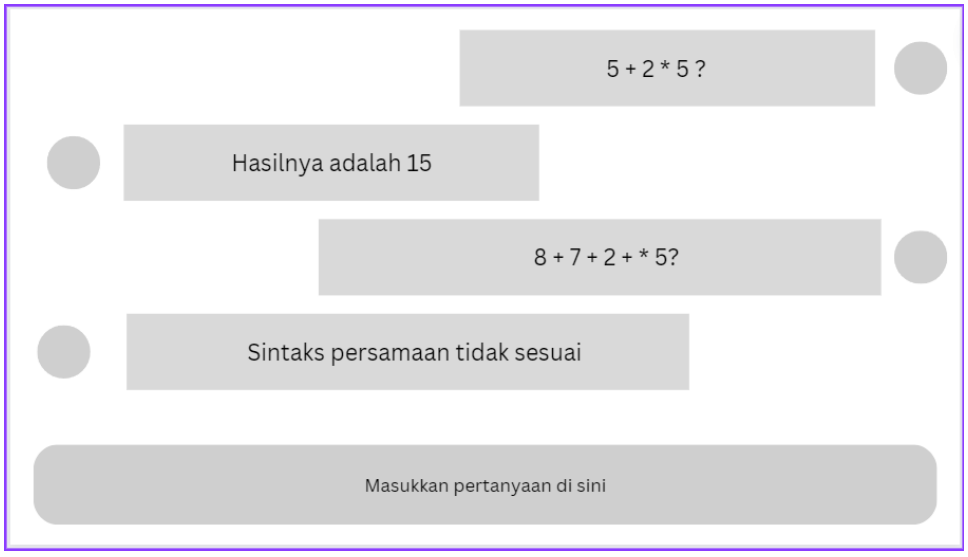
Klasifikasi dilakukan menggunakan regex dan terklasifikasi layaknya bahasa sehari - hari. Algoritma string matching KMP dan BM digunakan untuk klasifikasi *query* teks. Tersedia *toggle* untuk memilih algoritma KMP atau BM. Semua pemrosesan respons dilakukan pada sisi *backend*. Jika ada pertanyaan yang tidak sesuai dengan fitur, maka tampilkan saja "Pertanyaan tidak dapat diproses". Berikut adalah beberapa contoh ilustrasi sederhana untuk tiap pertanyaannya. (*Note*: Tidak wajib mengikuti ilustrasi ini, tampilan disamakan dengan chatGPT juga boleh)



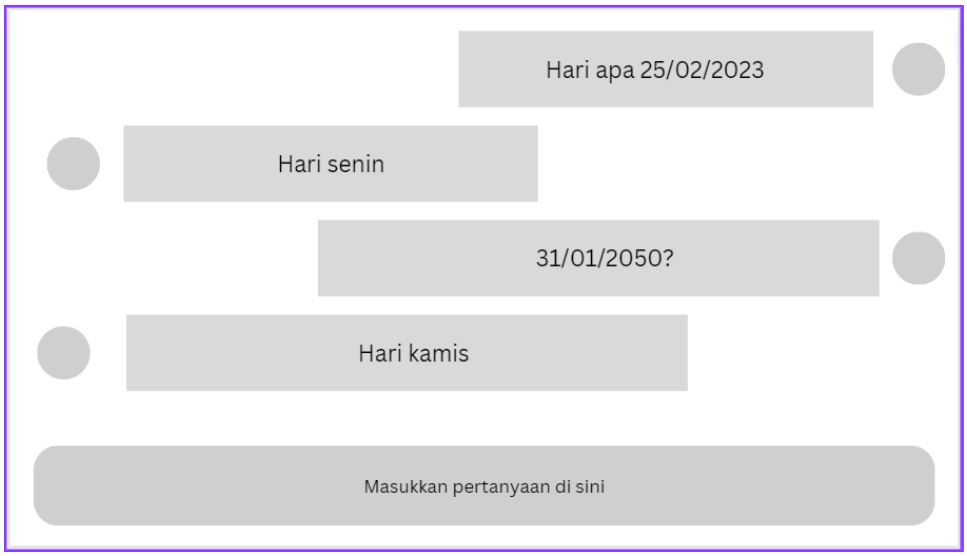
Gambar 1.2. Ilustrasi fitur pertanyaan teks kasus *exact*



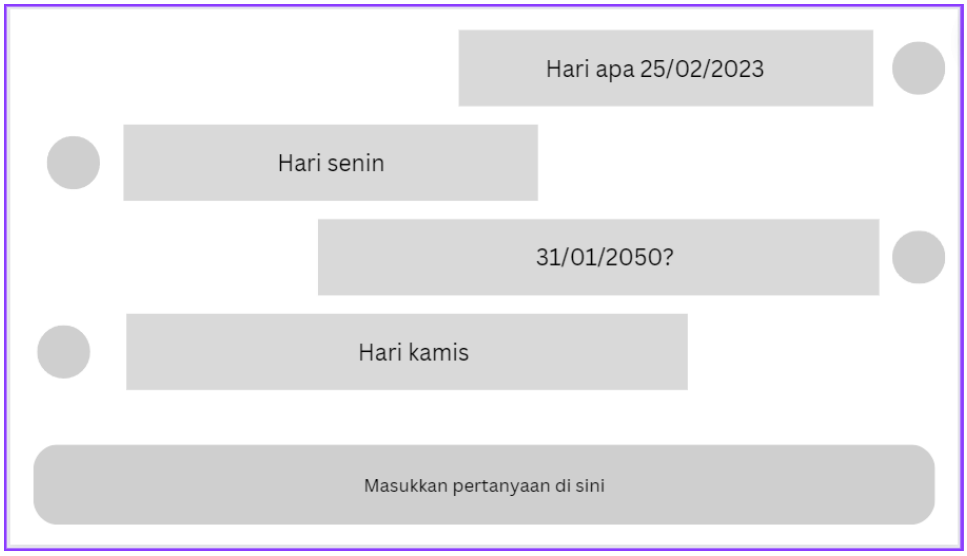
Gambar 1.3. Ilustrasi fitur pertanyaan teks kasus tidak *exact*



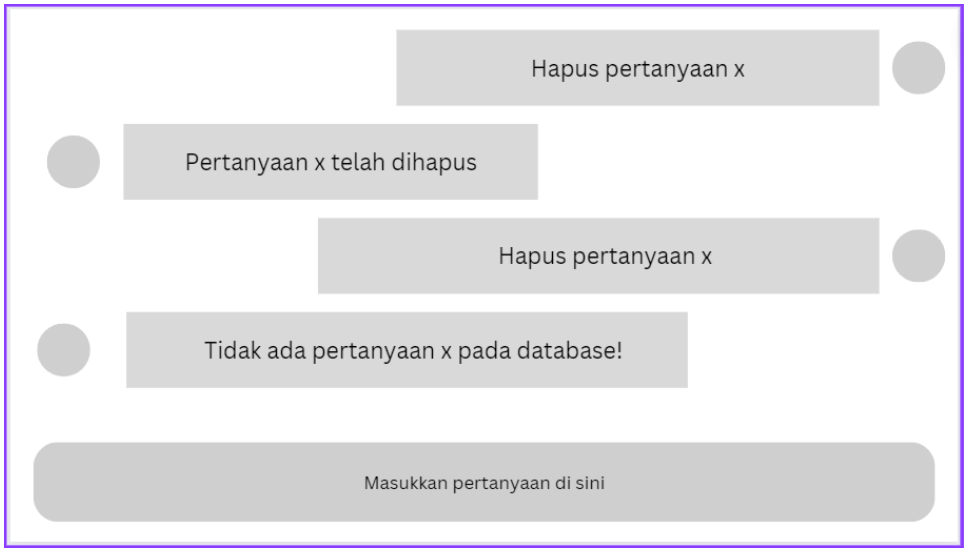
Gambar 1.4. Ilustrasi fitur kalkulator



Gambar 1.5. Ilustrasi fitur tanggal

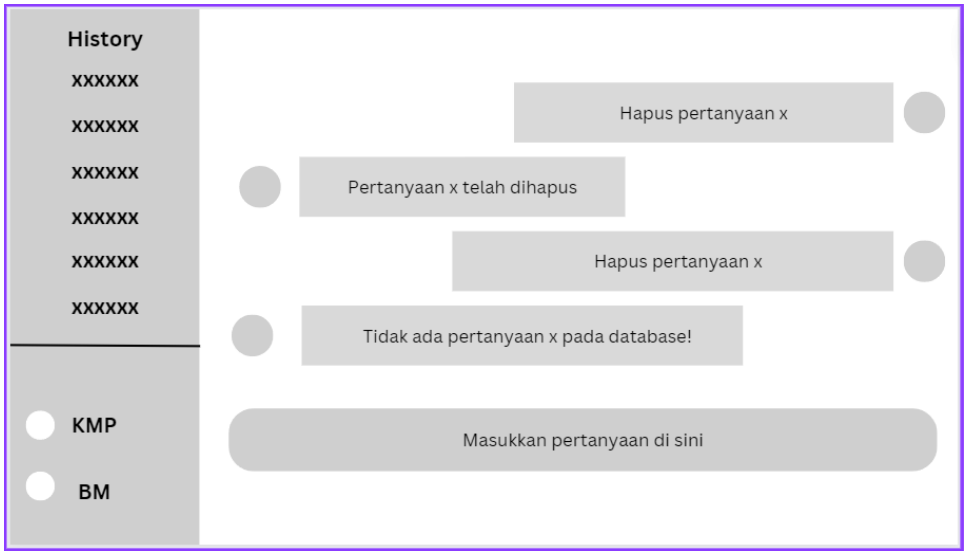


Gambar 1.6. Ilustrasi fitur tambah pertanyaan



Gambar 1.7. Ilustrasi Fitur Hapus Pertanyaan

Layaknya ChatGPT, di sebelah kiri disediakan *history* dari hasil pertanyaan anda. Cukup tampilkan 5-10 pertanyaan terbaru di *toolbar* kiri. Perhatikan bahwa sistem *history* disini disamakan dengan chatGPT, sehingga satu *history* yang diklik menyimpan seluruh pertanyaan pada sesi itu. Apabila *history* diclick, maka akan me-*restore* seluruh pertanyaan dan jawaban di halaman utama. Contoh ilustrasi keseluruhan:



Gambar 1.8. Ilustrasi Keseluruhan

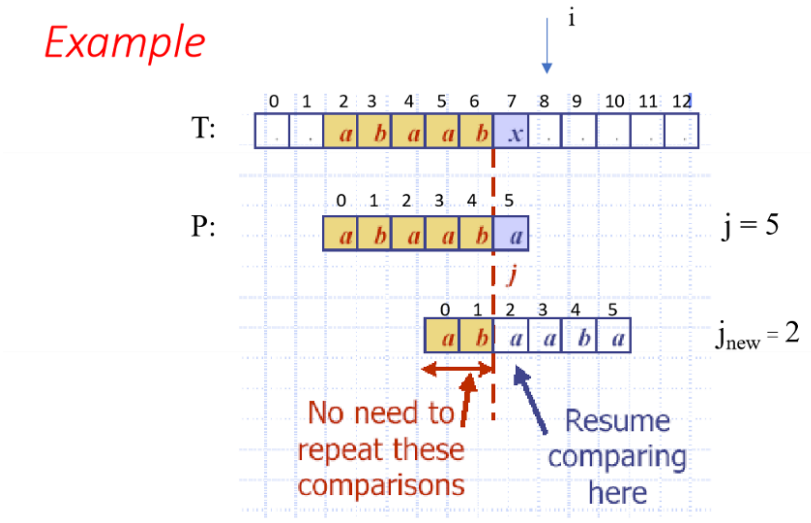
Spesifikasi Program:

1. Aplikasi berbasis *website* dengan pembagian Frontend dan Backend yang jelas.
2. Implementasi Backend wajib menggunakan Node.js / Golang, sedangkan Frontend dibebaskan tetapi disarankan untuk menggunakan React / Next.js / Vue / Angular. Lihat referensi untuk selengkapnya.
3. Penyimpanan data wajib menggunakan basis data (MySQL / PostgreSQL / MongoDB).
4. Algoritma pencocokan string (KMP dan Boyer-Moore) dan Regex wajib diimplementasikan pada sisi Backend aplikasi.
5. Informasi yang wajib disimpan pada basis data:
   a. Tabel pasangan pertanyaan dan Jawaban
   b. Tabel *history*
6. Skema basis data dibebaskan asalkan mencakup setidaknya kedua informasi di atas.
7. Proses *string matching* pada tugas ini **Tidak *case sensitive***.
8. Pencocokan yang dilakukan adalah dalam satu kesatuan string pertanyaan utuh (misal “Apa ibukota Filipina?”), bukan kata per kata (“apa”, “ibukota”, “Filipina”).

BAB II
LANDASAN TEORI

A. Algoritma Knuth–Morris–Pratt

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma *string / pattern matching* yang menyamakan pola dari kiri ke kanan (seperti algoritma *brute force*), tetapi pergeseran posisi pemeriksaan ketika pola tidak lagi sama tidak dilakukan secara satu per satu. Misalkan T adalah teks yang diperiksa dan P adalah pola yang dicari. Jika terjadi ketidaksamaan di karakter $T[i] \neq P[j]$, maka kita bisa menggeser posisi pencarian pada pola sejauh *prefix* terbesar $P[0 \ldots j-1]$ yang juga merupakan *suffix* $P[1 \ldots j-1]$ untuk menghindari perbandingan yang tidak diperlukan. Contohnya ada di gambar berikut :



Gambar 2.1. Pergeseran posisi pencarian pada algoritma Knuth-Morris-Pratt.

KMP memerlukan fungsi pinggiran (*border function*) yang menghitung ke posisi mana pencarian pola perlu digeser jika terjadi ketidaksamaan. Algoritma KMP dalam kode Java adalah sebagai berikut.

```
public static int kmpMatch (String text, String pattern) {
    int n = text.length();
    int m = pattern.length();
    int fail[] = computeFail(pattern);
    int i = 0; int j = 0;
```

```
    while (i < n) {
        if (pattern.charAt(j) == text.charAt(i)) {
            if (j == m - 1)
                return i - m + 1;  // match
            i++;
            j++;
        } else if (j > 0) {
            j = fail[j - 1];
        } else {
            i++;
        }
    }

    return -1;  // no match
}  // end of kmpMatch()


public static int[] computeFail (String pattern) {
    int fail[] = new int[pattern.length()];
    fail[0] = 0;
    int m = pattern.length();
    int j = 0; int i = 1;

    while (i < m) {
        if (pattern.charAt(j) == pattern.charAt(i)) { // j+1 chars match
            fail[i] = j + 1;
            i++;
            j++;
        } else if (j > 0) { // j follows matching prefix
            j = fail[j-1];
        } else {  // no match
            fail[i] = 0;
            i++;
        }
    }

    return fail;
} // end of computeFail()
```

Tabel 1. Algoritma Knuth-Morris-Pratt dalam bahasa Java.

Algoritma KMP memiliki kompleksitas waktu $O(m + n)$ dengan m menyatakan panjang pola dan n menyatakan panjang teks yang diperiksa. Kelebihan algoritma KMP adalah pencarian pada teks tidak perlu mundur, sehingga cocok untuk memproses *file* yang sangat besar. Kekurangannya adalah efektivitas KMP berkurang seiring jenis karakter bertambah.

B. Algoritma Boyer–Moore

Algoritma Boyer-Moore adalah algoritma *string / pattern matching* lain yang melakukan pencocokan dari belakang pola. Algoritma ini didasarkan pada 2 teknik, yaitu :

1. *Looking glass technique* :

   Mencari pola P di dalam teks T dengan menelusuri P secara mundur.

2. *Character jump technique* :

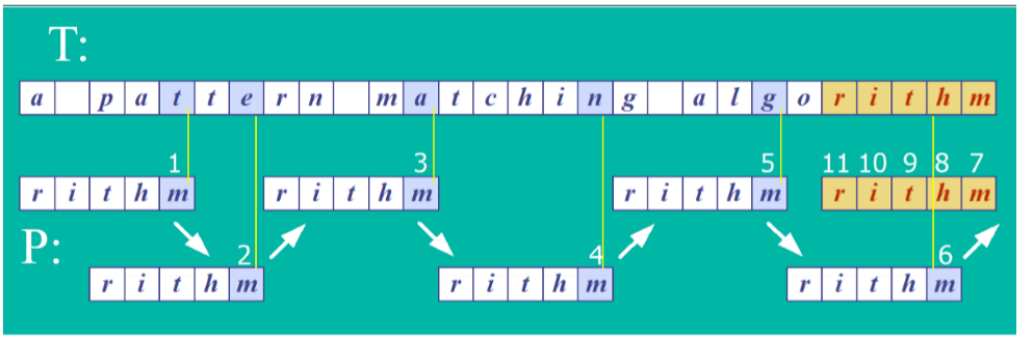
   Apabila $T[i] \neq P[j]$ (misal $T[i]$ merupakan karakter x), uji 3 kemungkinan berikut :

   a. Jika P mengandung x, geser P ke kanan sampai x terakhir di P ada di posisi bersesuaian dengan $T[i]$.

   b. Jika P mengandung x, tetapi pergeseran ke kanan tidak dapat dilakukan, geser P ke kanan sebanyak 1 sehingga pencarian dilanjutkan di posisi $T[i + 1]$.

   c. Jika kedua kasus di atas tidak ditemukan, geser P sehingga $P[0]$ bersesuaian dengan $T[i + 1]$.

Contoh algoritma Boyer-Moore adalah sebagai berikut :



Gambar 2.2. Contoh penggunaan algoritma Boyer-Moore.

Mirip seperti KMP, algoritma Boyer-Moore memerlukan fungsi kemunculan terakhir yang menunjukkan posisi terakhir kemunculan suatu karakter di teks T pada pola P. Algoritma Boyer-Moore dalam kode Java adalah sebagai berikut.

```
public static int bmMatch (String text, String pattern) {
    int last[] = buildLast(pattern);
    int n = text.length();
    int m = pattern.length();
```

```
    int i = m-1;
    if (i > n-1)
        return -1;  // no match if pattern is longer than text

    int j = m-1;
    do {
        if (pattern.charAt(j) == text.charAt(i)) {
            if (j == 0) {
                return i;  // match
            } else {  // looking-glass technique
                i--;
                j--;
            }
        } else {  // character jump technique
            int lo = last[text.charAt(i)];  //last occ
            i = i + m - Math.min(j, 1+lo);
            j = m - 1;
        }
    } while (i <= n-1);

    return -1;  // no match
}  // end of bmMatch()


public static int[] buildLast (String pattern) {
    /* Return array storing index of last
        occurrence of each ASCII char in pattern. */

    int last[] = new int[128];  // ASCII charset
    for(int i = 0; i < 128; i++) {
        last[i] = -1; // initialize array
    }

    for (int i = 0; i < pattern.length(); i++) {
        last[pattern.charAt(i)] = i;
    }

    return last;
}  // end of buildLast()
```

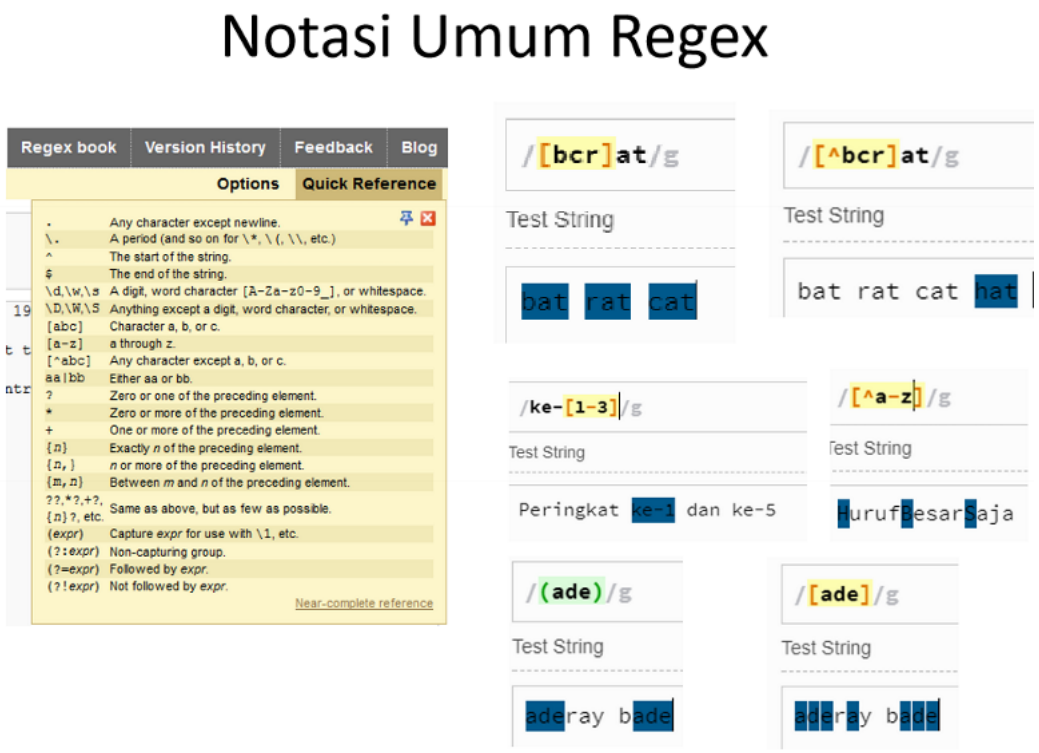
Tabel 2. Algoritma Boyer-Moore dalam bahasa Java.

Algoritma Boyer-Moore memiliki kompleksitas waktu $O(nm + A)$ dengan A adalah alfabet terkait. Algoritma ini cepat untuk alfabet yang karakternya banyak, namun lambat untuk alfabet yang karakternya sedikit.

C. *Regular Expression*

Apabila algoritma KMP dan Boyer-Moore hanya bisa mencari *exact match* dalam suatu pola, *regular expression* (Regex) bisa mencocokkan *string* manapun yang memenuhi

aturan pola tertentu. *Regular expression* adalah sekumpulan notasi dan karakter yang digunakan untuk mendeskripsikan suatu pola pada pencarian berbasis karakter. Biasanya regex digunakan pada operasi "*find*" atau "*find and replace*" pada *string*. Regex didefinisikan dengan sintaks khusus, contohnnya seperti di bawah ini :



Gambar 2.3. Beberapa notasi umum regex.

Selain digunakan untuk mendeskripsikan sebuah pola, regex juga dapat digunakan untuk melakukan validasi terhadap suatu masukan tertentu. Misalkan terdapat sebuah *regular expression* yang digunakan untuk melakukan validasi email masukan sebagai berikut :

```
^[a-zA-Z0-9+_.-]+@[a-zA-Z0-9.-]+$
```

Maka "example.samplemail@gmail.com" akan menjadi email yang valid dan "sample?examplemail@gmail.com" akan menjadi email yang tidak valid karena tidak memenuhi aturan regex yang didefinisikan.

D. *Longest Common Subsequence* (LCS)

Longest Common Subsequence (LCS) adalah algoritma yang mencari *subsequence* terpanjang (tidak harus karakter yang bersebelahan, namun tetap terurut dari kiri ke kanan) dari dua buah *string*. Misal ada dua string berikut :

S1 = "AGGTAB"

S2 = "GXTXAYB"

maka *longest common subsequence* dari S1 dan S2 adalah "GTAB" dengan panjang 4.

Algoritma LCS bisa diimplementasikan dengan beberapa cara, salah satunya adalah algoritma yang menggunakan *memoization* seperti pada kode berikut :

```
/* A Top-Down DP implementation of LCS problem */
/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
function lcs (X, Y, m, n, dp) {
    if (m == 0 || n == 0)
        return 0;

    if (X[m - 1] == Y[n - 1])
        return dp[m][n] = 1 + lcs(X, Y, m - 1, n - 1, dp);

    if (dp[m][n] != -1) {
        return dp[m][n];
    }

    return dp[m][n] = Math.max(lcs(X, Y, m, n - 1, dp),
                        lcs(X, Y, m - 1, n, dp));
}
```

Tabel 3. Algoritma LCS menggunakan *memoization* dalam bahasa Javascript.

Implementasi di atas memiliki kompleksitas waktu $O(mn)$ dan kompleksitas ruang $O(mn)$ untuk penyimpanan hasil masing-masing tahap.

E. Pembangunan Aplikasi Web

Aplikasi web dibangun secara dominan menggunakan bahasa pemrograman javascript dan MongoDB untuk mendukung sistem basis data, menyimpan daftar pertanyaan yang sudah pernah ditanyakan sebelumnya dan jawaban atas pertanyaan-pertanyaan yang disampaikan dari masukan pengguna. Algoritma pencocokan *string* seperti algoritma KMP dan BM yang telah dibahas pada bagian sebelumnya dilakukan pada sisi *backend*. Digunakan pula *regular expression* untuk melakukan klasifikasi terhadap setiap *query* pertanyaan yang

masuk ke dalam setiap kategori yang ada pada aplikasi web yang dibangun. Selain itu, terdapat pula algoritma yang digunakan untuk melakukan kalkulasi kemiripan pertanyaan dengan algoritma *Longest Common Subsequence*.

Aplikasi web yang dibangun dapat melakukan beberapa fitur, diantaranya sistem *login* yang memungkinkan setiap pengguna memiliki data *chat* aktif maupun *history* yang berbeda. Fitur utama dari aplikasi ini adalah untuk melaksanakan sistem *chatbot* yang dapat melakukan tanya-jawab (*question-answering*) berdasarkan pertanyaan yang sudah ada pada database, maupun beberapa fitur yang dapat dilakukan oleh *chatbot*, seperti melakukan kalkulasi matematis berdasarkan *query* yang diterima, menentukan hari dari tanggal masukan, menghapus dan menambahkan pertanyaan, dan lain sebagainya. Selain itu, aplikasi juga mendukung sistem *history* yang memungkinkan pengguna untuk dapat melihat daftar pertanyaan yang dilakukan pada sesi dan waktu tertentu dan dapat membukanya kembali. Pengguna juga dapat memberi nama terhadap sebuah laman jika dirasa perlu, melakukan regenerasi jawaban, melakukan penyuntingan pertanyaan (dikirimkan sebagai pertanyaan yang baru), dan menyalin jawaban ke dalam *clipboard* sehingga dapat disalin kembali pada tempat yang berbeda tanpa harus melakukannya secara manual (cukup tekan satu tombol "*copy*" saja).

BAB III
ANALISIS PEMECAHAN MASALAH

A. Langkah - langkah Pemecahan Masalah

Berikut adalah beberapa langkah-langkah pemecahan masalah yang dilakukan selama proses pembangunan aplikasi berbasis web.

1. Pemecahan Masalah Fitur *Question-Answering* (QA)

Aplikasi akan meminta masukan dari pengguna melalui sebuah kotak masukan yang terdapat pada bagian bawah aplikasi. Bagian ini berada pada sisi *frontend* sehingga pengguna dapat berhubungan langsung dengan kotak masukan ini. Pengguna bebas memberikan masukan pertanyaan dalam bentuk apapun sesuai dengan spesifikasi fitur yang telah dijelaskan pada bagian spesifikasi. Pengguna dapat memberikan *query* masukan lebih dari satu (misal melakukan fitur kalkulator dan menambahkan pertanyaan) pada saat yang bersamaan dengan menggunakan karakter ";" sebagai perantaranya. Pengguna juga dapat memilih algoritma pencocokan *string* yang hendak digunakan untuk melakukan pemrosesan jawaban berdasarkan masukan yang telah diberikan. Pada saat pengguna mengirimkan jawabannya dengan menekan tombol "enter" atau tombol *send* bergambar pesawat pada aplikasi, keseluruhan pertanyaan yang menjadi masukan dari pengguna akan disimpan dalam sebuah struktur data pesan. Di saat yang sama pertanyaan tersebut akan dikirimkan ke pihak *backend* untuk kemudian dilakukan pemrosesan jawaban.

Ketika menerima sebuah pertanyaan, *backend* akan melakukan klasifikasi jenis pertanyaan menggunakan regex terlebih dahulu. Berikut adalah klasifikasi skema pemrosesan jawaban berdasarkan fitur yang ada :

a. Apabila pertanyaan termasuk kategori tanggal, maka akan dikembalikan hari pada tanggal tersebut.
b. Apabila merupakan persamaan matematika, layaknya kalkulator, maka akan dikembalikan hasilnya.
c. Apabila merupakan permintaan penambahan pertanyaan dan jawaban, maka pertanyaan dan jawaban tersebut akan ditambahkan ke *database*.

d. Apabila merupakan permintaan penghapusan pertanyaan, maka pertanyaan akan dihapus dari *database*.
e. Apabila tidak termasuk dalam kategori-kategori yang telah dijelaskan diatas, maka pertanyaan akan langsung dianggap sebagai pertanyaan biasa dan akan dilakukan pencarian pertanyaan di *database* dengan menggunakan algoritma Boyer-Moore / KMP (sesuai permintaan). Apabila tidak ditemukan pertanyaan yang sama persis, pencarian dilakukan dengan *longest common subsequence* dengan tingkat kemiripan minimal 90%. Apabila pertanyaan yang sesuai ditemukan, maka jawaban dari pertanyaan tersebut akan dikembalikan sebagai *response*. Jika tidak, maka akan dikembalikan "Pertanyaan tidak ada di database". Jika ada pertanyaan yang kemiripannya > 40% namun < 90%, maka akan dituliskan sebagai tebakan pertanyaan yang dimaksud.

Setelah jawaban berhasil didapatkan dari sisi *backend*, hasil akan dikirimkan kembali kepada sisi *frontend* dalam bentuk sebuah larik (*list*) jawaban. Alasan dipilihnya penggunaan larik adalah untuk memudahkan pengembalian hasil untuk *query* masukan yang mungkin lebih dari satu. Setelah jawaban diterima oleh sisi *frontend,* maka akan dilakukan penampilan jawaban hail pemrosesan *query* masukan pada layar, tepat di bawah posisi masukan sebelumnya diberikan. Hasil pemrosesan jawaban dapat berupa apapun dan pengguna dapat menerima respon atas *query* yang diberikan termasuk jika pengguna memberikan lebih dari satu pertanyaan pada saat yang bersamaan.

Lebih lanjut, setiap jawaban yang mewakili pertanyaan akan kembali disimpan dalam struktur data pesan, lengkap dengan statusnya dan untuk kemudian dapat disimpan dalam basis data untuk id pengguna terkait. Hal ini dilakukan agar jika pengguna melakukan pemuatan ulang (*reload*), pengguna tetap dapat menyimpan daftar pertanyaan yang sebelumnya sudah ditanyakan sebelumnya.

2. Pemecahan Masalah Fitur *History*

Fitur *history* dapat diimplementasikan dengan baik dengan menggunakan struktur data pesan yang telah banyak disinggung pada bagian sebelumnya. Secara umum struktur data sebuah halaman terdiri atas nama dari halaman terkait dan struktur data pesan yang disimpan dalam halaman tersebut. Adapun sebuah struktur

data pesan akan menyimpan sebuah pertanyaan dalam bentuk *string,* status dari pertanyaan (apakah sudah terjawab atau belum), dan jawaban atas pertanyaan yang menjadi masukan. Struktur data ini kemudian dibuat dan dimodifikasi secara berkala seiring perlakuan pengguna terhadap aplikasi sehingga perlu untuk dilakukan pembaharuan terhadap data halaman secara berkala. Data baru ini akan dikirim ke *backend*.

Setiap kali menerima data *history* baru, *backend* akan mengakses *database* untuk mengupdate *history user* yang bersangkutan. Apabila *user* merupakan pengguna baru, maka akan dibuat *document* baru di *database*. Apabila *user* sudah terdaftar, maka *history* untuk *user* tersebut akan dicari (berdasarkan ID) dan diperbaharui dengan *chat history* yang baru. Saat *user* pertama membuka *website*, akan dilakukan pencarian data *history user* tersebut. Apabila ditemukan, *chat* yang tersimpan akan dimunculkan. Bila tidak, *chat* akan tetap kosong.

3. **Pemecahan Masalah Implementasi Basis Data**

   Implementasi fitur penyimpanan menggunakan *database* dilakukan dengan menggunakan MongoDB sebagai DBMS *website*. Terdapat 2 buah model data yang diimplementasikan, satu untuk penyimpanan pertanyaan dan jawabannya (model QuestionSchema), serta satunya untuk menyimpan *history* pertanyaan dan jawaban untuk tiap pengguna (model HistorySchema).

4. **Pemecahan Masalah *Deployment***

   Deployment aplikasi memanfaatkan layanan AWS, terutama layanan EC2 *instance*. Layanan tersebut memungkinkan pengembang untuk menjalankan aplikasi (baik berupa arsitektur *frontend* maupun *backend*) di dalam *instance* tersebut yang dapat berjalan secara terus menerus di *cloud*. Digunakan nama domain yang dimiliki oleh salah satu pengembang sebagai domain aplikasi dalam *deployment*. Agar aplikasi tersebut dapat tersambung ke domain yang ingin digunakan, digunakan juga layanan *Elastic IP Address* agar menjaga konsistensi *IP address* dari *instance* yang dipakai. Dengan demikian, domain yang dimiliki dapat dihubungkan ke IP *address* tersebut.

   Karena menggunakan nama domain *custom*, perlu dikonfigurasi Nginx sebagai *reverse proxy* pada *instance* yang digunakan. *Port* 3000 pada *localhost*

kemudian terhubung ke alamat relatif “/” yang terhubung ke *frontend* aplikasi sedangkan *Port* 5000 terhubung ke alamat relatif “/api” yang terhubung ke *backend* aplikasi. Selain itu, karena aplikasi memanfaatkan sistem autektikasi *user*, API yang ingin digunakan mengharuskan penggunaan protokol *https*. Dalam aplikasi ini, Certbot digunakan dalam mendapatkan SSL *certificate*.

B. Fitur Fungsional Aplikasi

Berdasarkan beberapa langkah-langkah yang dilakukan untuk menyelesaikan masalah yang ada pada spesifikasi tugas besar, berikut adalah beberapa fitur fungsional yang diimplementasikan pada aplikasi web yang dibangun.

1. **Fitur *Question-Answering* (QA)**

   Seperti yang sudah disampaikan sebelumnya bahwa pengguna dapat memberikan masukan berupa sebuah *query* yang dapat dikirimkan melalui kotak masukan yang tersedia. Pengguna dapat memberikan beberapa perintah dalam sebuah *query* yang sama dengan menggunakan perantara karakter “;” . Berikut adalah beberapa fitur yang dapat ditangani oleh aplikasi dan contoh *query* yang dapat digunakan.

   a. **Fitur Cari Hari Berdasarkan Tanggal**

      Pengguna dapat mengetikkan "[hari apa] DD/MM/YYYY" untuk mendapatkan hari pada tanggal tersebut. Pengguna juga dapat hanya mengetikkan “DD/MM/YYYY” dan memperoleh hasil yang sama untuk tanggal terkait.

   b. **Fitur Kalkulator**

      Pengguna dapat memberi masukan persamaan matematika untuk mendapatkan hasil dari persamaan tersebut. Adapun operasi matematis dibatasi pada operasi tambah, kurang, kali, bagi, pangkat, kurung, dan operasi negatif.

   c. **Fitur Tambah Pertanyaan**

      Pengguna dapat menambah pernyataan dengan mengetik "Tambah[kan] pertanyaan <pertanyaan> dengan jawaban <jawaban>". Pertanyaan dan jawaban yang menjadi masukan dari pengguna akan disimpan ke dalam *database* global sehingga dapat pula diakses oleh setiap pengguna.

d. Fitur Hapus Pertanyaan

Pengguna dapat memberikan masukan "Hapus pertanyaan <pertanyaan>" untuk menghapus sebuah pertanyaan dari *database* dan pertanyaan akan dihapus secara global sehingga setiap pengguna yang terdaftar tidak akan dapat memperoleh jawaban dari pertanyaan tersebut.

e. Fitur Jawaban

Pengguna dapat mengetikkan sebuah pertanyaan dan program akan melakukan pencarian jawaban terhadap pertanyaan tersebut berdasarkan daftar pertanyaan dan jawaban yang terdapat di dalam *database*. Pencarian pertanyaan dapat dilakukan dengan menggunakan algoritma KMP/Boyer-Moore berdasarkan pilihan dari pengguna pada tombol yang tersedia.

2. Fitur History

Aplikasi dapat menyimpan semua pertanyaan dan jawaban yang pernah disampaikan oleh pengguna. Fitur ini membuat pengguna dapat membuka atau melihat kembali daftar pertanyaan dan jawaban yang pernah ditanyakan dalam sebuah sesi pertanyaan tertentu. Jumlah percakapan yang dapat disimpan hingga 20 halaman sehingga pengguna tidak perlu khawatir terkait lupa mengenai pertanyaan yang pernah ditanyakan sebelumnya.

3. Fitur Tambahan Aplikasi

Selain berbagai fitur yang disampaikan pada spesifikasi, pada bagian *frontend* juga diimplementasikan beberapa fitur yang dapat membantu pengguna agar dapat menggunakan aplikasi web secara lebih mudah dan praktis. Berikut adalah penjelasannya.

a. Fitur *Copy*

Fitur ini memungkinkan pengguna untuk menyalin jawaban yang diberikan oleh aplikasi untuk kemudian dapat dituliskan kembali pada destinasi yang lain. Fitur ini dapat digunakan dengan menekan tombol "*copy*" pada bagian bawah jawaban sebuah *chat* tertentu.

b. Fitur *Regenerate Response*

Fitur ini akan membuat jawaban sebelumnya diperbaharui menjadi jawaban yang baru. Meskipun seringkali hasil tidak mengalami banyak perubahan

karena menggunakan mekanisme yang sama, tetapi tombol ini dapat membawa pengguna kepada jawaban yang lebih akurat dan lebih dibutuhkan.

c. Fitur *Edit Question*

Pengguna yang merasa mengajukan pertanyaan yang kurang tepat atau terpotong karena telah telanjur mengetikkan tombol "enter" pada keyboard atau *send* pada aplikasi dapat melakukan perubahan dari pertanyaan yang ada dan pertanyaan tersebut akan menjadi pertanyaan baru yang siap menerima jawaban yang baru.

d. Fitur *Delete Chat*

Layaknya ChatGPT, halaman *chat* yang dirasa sudah tidak diperlukan lagi dapat dihapus sesuai keinginan pengguna dengan menekan tombol *trash can* pada samping kanan setiap halaman *chat*. Fitur ini akan membuat organisasi halaman dari pengguna menjadi lebih terstruktur.

e. Fitur *Login* dan *Logout*

Setiap pengguna akan memiliki sebuah laman dan halaman *history* yang berbeda. Tentu saja hal ini dapat direalisasikan dengan adanya sebuah mekanisme autentikasi pengguna melalui fitur *login* dan *logout*. Halaman yang akan dimuat pada aplikasi adalah daftar halaman yang sebelumnya pernah diakses dan dibuat oleh pengguna tersebut. Jika pengguna merupakan pengguna baru yang baru saja terdaftar, maka akan ditampilkan halaman kosong yang siap untuk diisi dengan dialog kepada *chatbot*. Fitur ini didukung oleh autentikator auth0 yang relatif mudah untuk digunakan dan diimplementasikan.

C. Arsitektur Aplikasi Web

Aplikasi dari sisi *frontend* dibangun dengan menggunakan bahasa pemrograman Javascript dengan *framework* React.js. Sementara itu untuk melakukan penggayaan (*styling*) digunakan *framework* Tailwind CSS sehingga bisa mendefinisikan penggayaan secara *inline*. Terdapat komponen tambahan yang digunakan untuk membantu memperindah tampilan seperti toastify untuk memunculkan pesan berhasil atau peringatan. Selain itu, aplikasi dibangun dengan tampilan yang responsif di berbagai *device*. Komunikasi dengan sisi *backend* dilakukan menggunakan beberapa fungsi perantara untuk melakukan proses *fetch*.

Aplikasi *backend* dibangun dengan bahasa pemrograman yang sama dan memanfaatkan *framework* Express. Database yang digunakan memanfaatkan layanan MongoDB. Untuk menyesuaikan skema model database yang digunakan pada *server*, digunakan *framework* Mongoose. Setiap API *backend* diproteksi dengan autentikasi token memanfaatkan JWT. Dengan demikian, API hanya dapat digunakan oleh *user* yang terautentikasi pada arsitektur *frontend*.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi Teknis Program

1. Struktur data

a. Skema model penyimpanan data pada *database*

1. **QuestionSchema** : untuk menyimpan pertanyaan dan jawabannya

```
const QuestionSchema = new mongoose.Schema({

    question : {
        type : String,
        require : true,
        unique : true,
    },

    answer : {
        type : Array,
        default : [],
        require : true,
        unique : false,
    },

});
```

Tabel 4. Skema penyimpanan pertanyaan dan jawabannya

2. **HistorySchema** :untuk menyimpan sejarah *chat* masing-masing pengguna

```
const HistorySchema = new mongoose.Schema({

    user_id : {
        type : String,
        require : true,
        unique : true,
    },

    pages : [{
        convo : [{
            question : String,
            answer : {
                type: Array,
                default: []
            },
            answered : Boolean
        }],
        name : {
            type : String,
```

```
            require : true,
            unique: false
        }
    }],

});
```

Tabel 5. Skema penyimpanan *chat history user*

b. Struktur data pesan (*convo*)

```
convo = [..., { question: String, answer: [...], answered:
boolean }, ...]
```

Tabel 6. Struktur data pesan

Pesan (*convo*) adalah sebuah larik berisi objek kompak berelemen pertanyaan (*question*) dalam bentuk String, jawaban (*answer*) yang dinyatakan juga sebagai sebuah larik, dan status jawaban (*answered*) dalam bentuk boolean untuk melakukan validasi dalam memunculkan kolom jawaban.

c. Struktur data halaman (*pages*)

```
pages = [..., { convo : [...], name : String }, ...]
```

Tabel 7. Struktur data halaman

Halaman (*pages*) juga merupakan sebuah larik yang berisi kumpulan halaman-halaman yang menyimpan *history* dari pengguna dengan id tertentu. Pesan terdiri atas objek kompak berelemen pesan (*convo*) yang telah dijelaskan sebelumnya dan nama dari halaman tersebut (*name*) dengan nilai standar adalah nama halaman diikuti dengan indeksnya.

2. Fungsi dan Prosedur

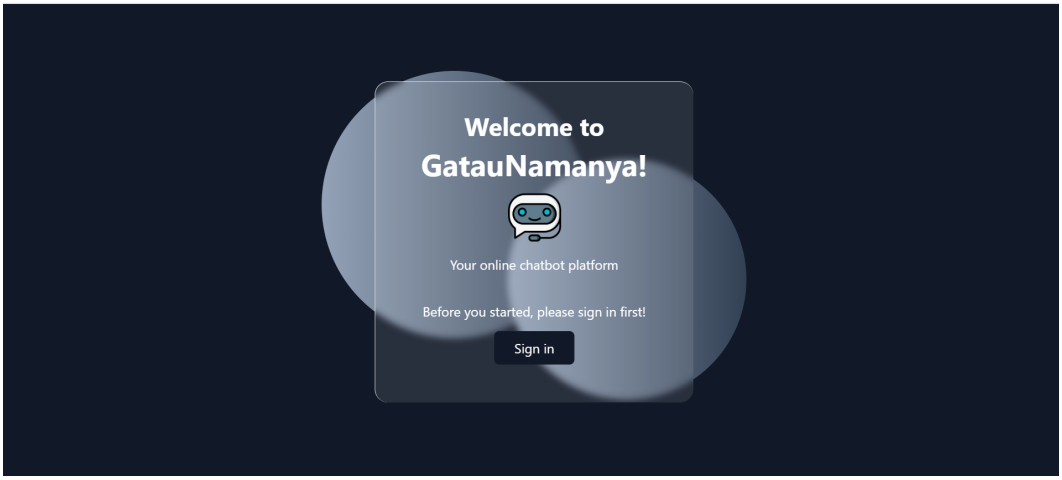
No	Nama Fungsi/Prosedur	Keterangan
1.	`booyer_moore(text,pattern)`	Mengembalikan indeks awal ditemukannya pattern dalam text menggunakan algoritma Boyer-Moore.
2.	`fill_last_indexes(pattern)`	Mengembalikan *dictionary* indeks terakhir kemunculan masing-masing karakter dalam pattern.
3.	`KMP (text,pattern)`	Mengembalikan indeks awal ditemukannya pattern dalam text

		menggunakan algoritma Knuth-Morris-Pratt.
4.	`getBorders(pattern)`	Mengembalikan *array* lokasi pergeseran jika terjadi *mismatch* di posisi tertentu.
5.	`LCS (firstText, secondText)`	mengembalikan panjang *common subsequence* yang terpanjang antara firstText dan secondText.
6.	`LCSrecursion (firstText, secondText, i, j, memo)`	Implementasi algoritma LCS menggunakan *memoization*.
7.	`identify_statement_type(text)`	Menentukan tipe pertanyaan/permintaan menggunakan regex.
8.	`getAnswer(text, algorithm)`	Mengembalikan jawaban pertanyaan yang ada di text dengan algoritma pencarian yang dipilih. Menangani fungsi kalkulator dan tanggal juga.
9.	`calculate(equation)`	Mengevaluasi ekspresi *infix* dan mengembalikan hasilnya.
10.	`process(operands,operators)`	Mengembalikan hasil operasi 2 operan teratas memakai operator teratas dari *stack* operands dan operators.
11.	`find_question(q, algorithm)`	Mengembalikan id pertanyaan pada *database* yang sama dengan / mirip q dengan algoritma yang dipilih.
12.	`delete_question(q, algorithm)`	Menghapus pertanyaan q dari *database* apabila ada. Pencarian pertanyaan dilakukan dengan algoritma yang diminta.
13.	`save_question(q,a,algorithm)`	Menyimpan pertanyaan q dengan jawaban a pada *database* jika belum ada. Jika pertanyaan sudah ada, jawaban pertanyaan akan diganti dengan a. Pencarian pertanyaan dilakukan dengan algoritma yang dipilih.
14.	`get_answer(q, algorithm)`	Mengembalikan jawaban untuk pertanyaan q dari *database* jika ada. Pencarian dilakukan dengan algoritma yang dipilih.

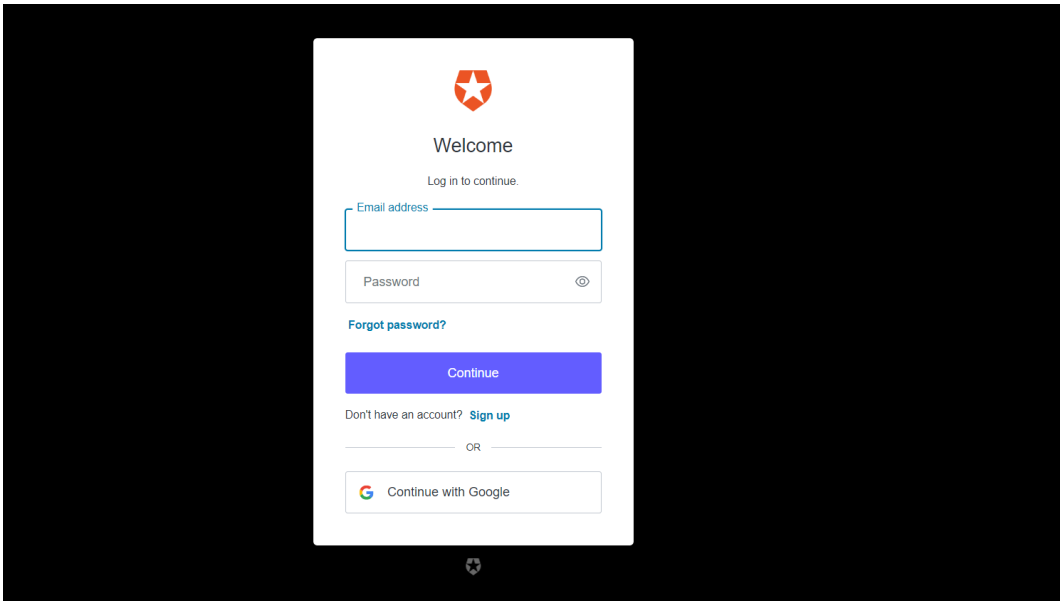
Tabel 8. Fungsi dan prosedur yang digunakan pada *backend*

B. Tata Cara Penggunaan Program

1. Login ke website dengan akun Auth0 atau Google.

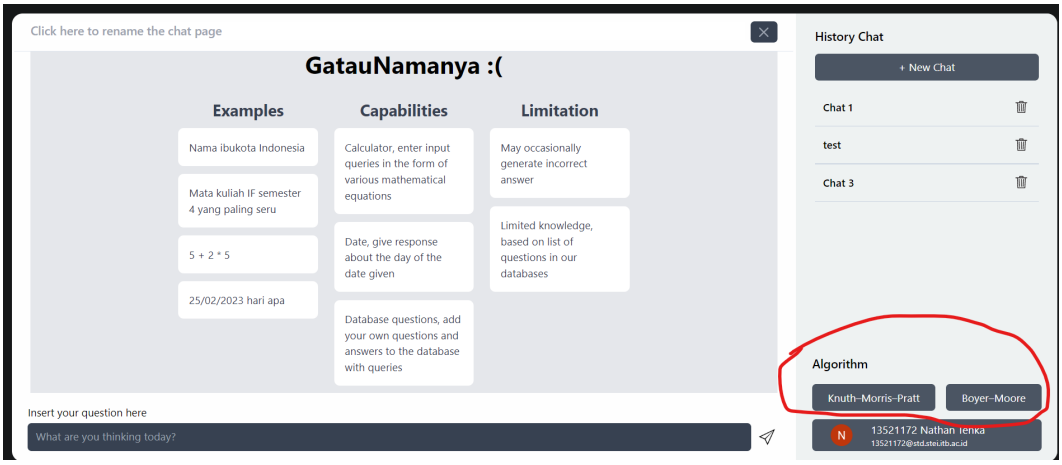


Gambar 4.1. Halaman awal saat membuka *website*.



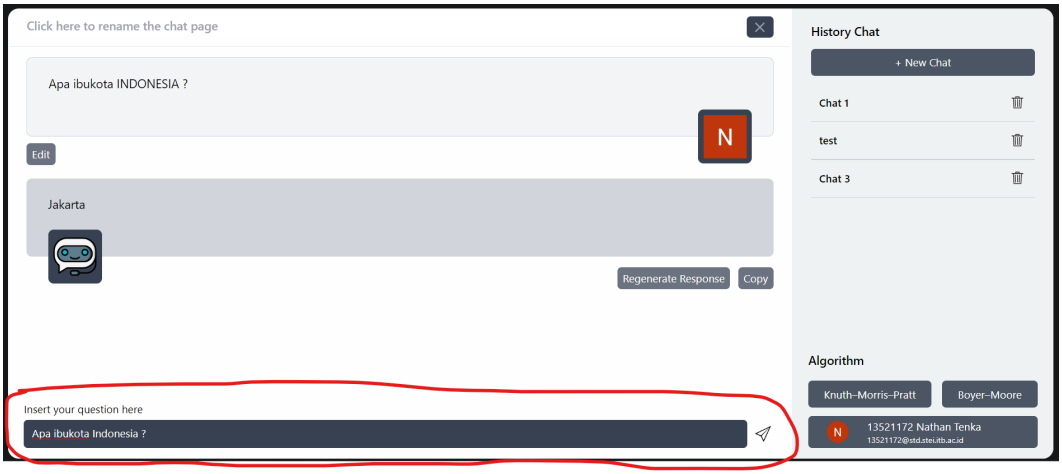
Gambar 4.2. Halaman *sign in*.

2. Pilih algoritma pencarian yang diinginkan di bawah kanan.



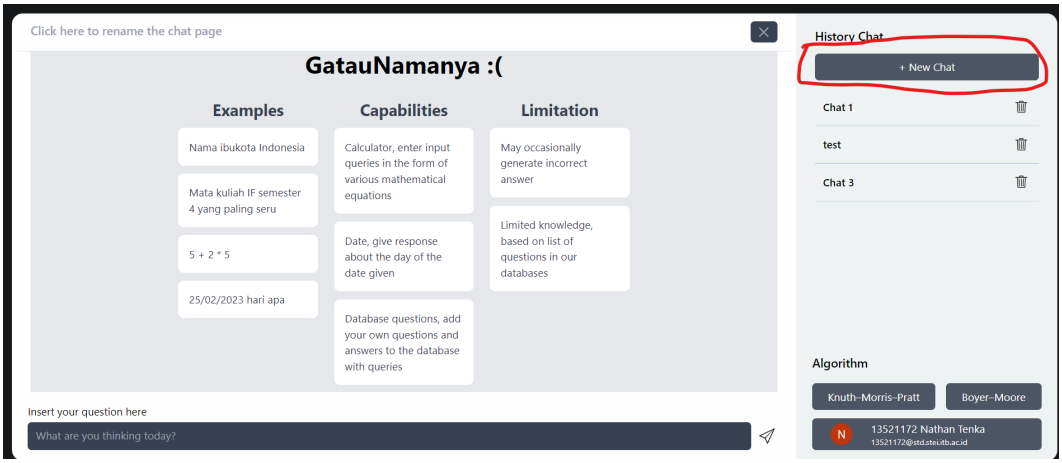
Gambar 4.3. Halaman *chat* dengan tombol pilihan algoritma ditandai.

3. Ketikkan pertanyaan ke kotak yang tersedia.



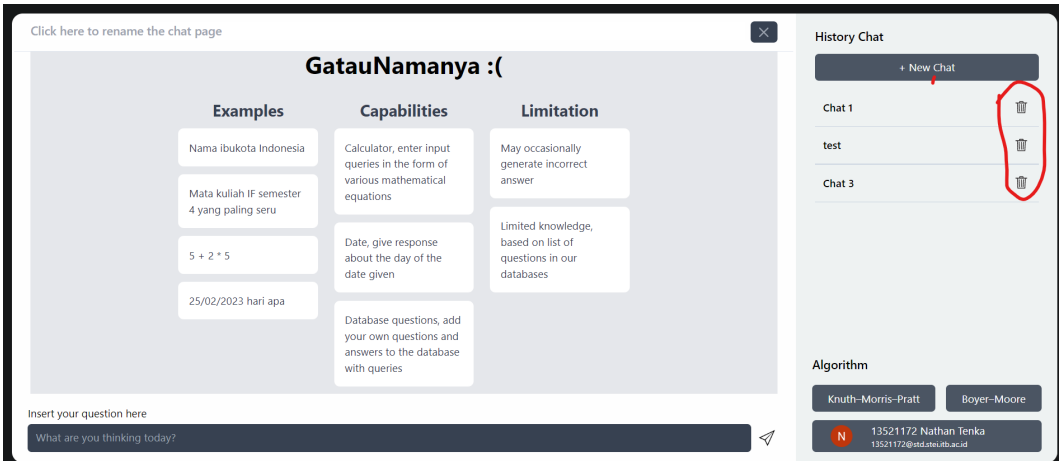
Gambar 4.4. Kotak pertanyaan.

4. Untuk menambahkan *chat* baru, klik tombol New Chat + di sebelah kanan atas.



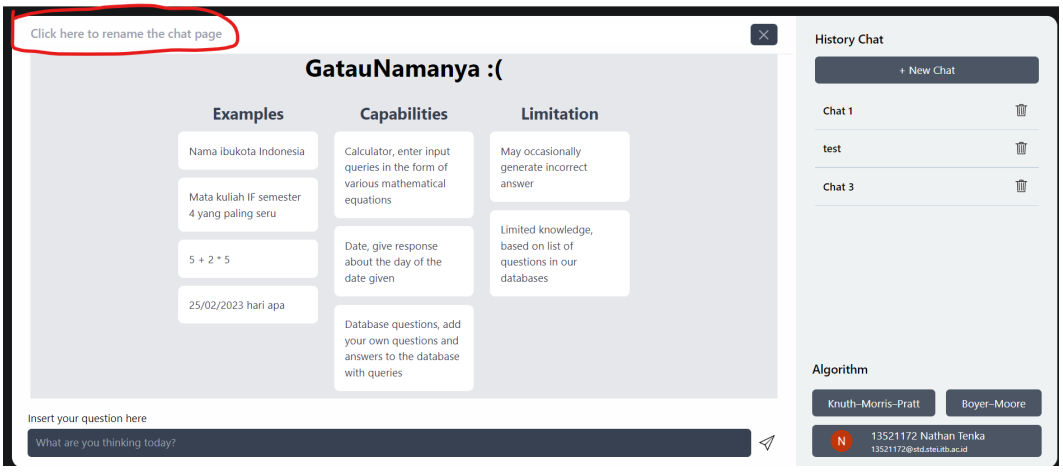
Gambar 4.5. Tombol tambah *chat* baru.

5. Untuk menghapus *chat*, klik tombol tempat sampah di sebelah kanan *chat* yang ingin dihapus.



Gambar 4.6. Tombol hapus *chat*.

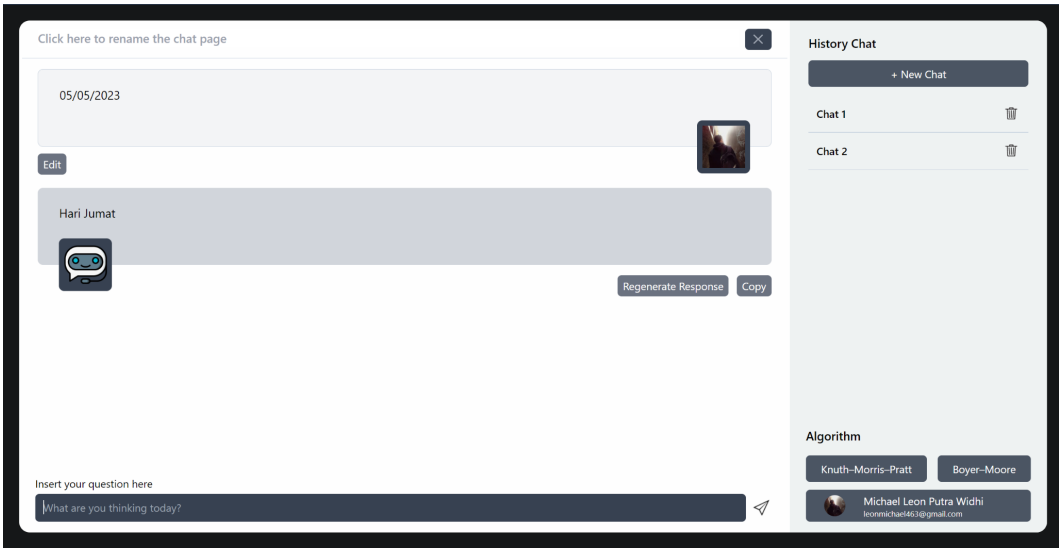
6. Untuk menamai ulang *chat*, klik tempat yang tersedia di sebelah atas.



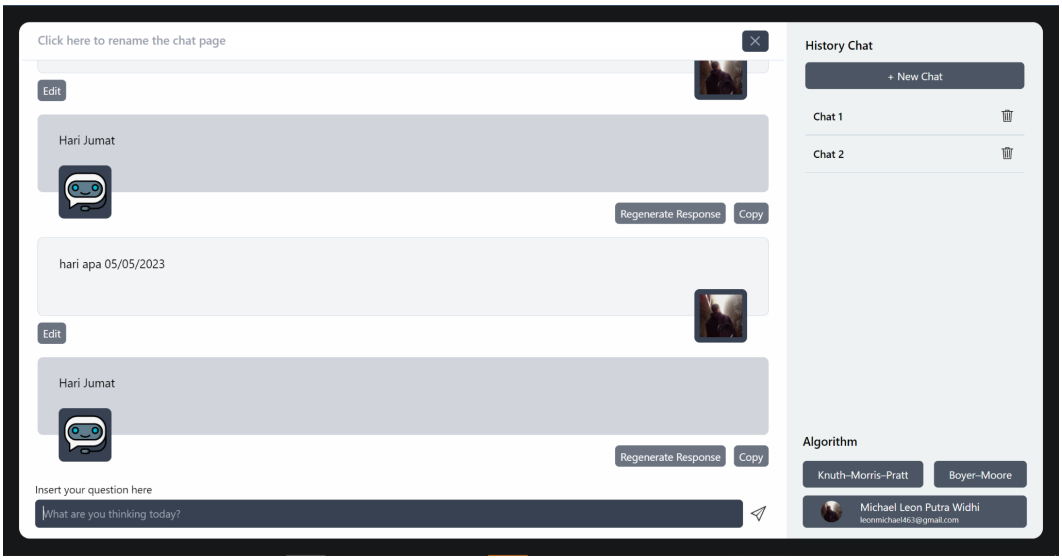
Gambar 4.7. Kotak *rename chat*.

C. Hasil Pengujian

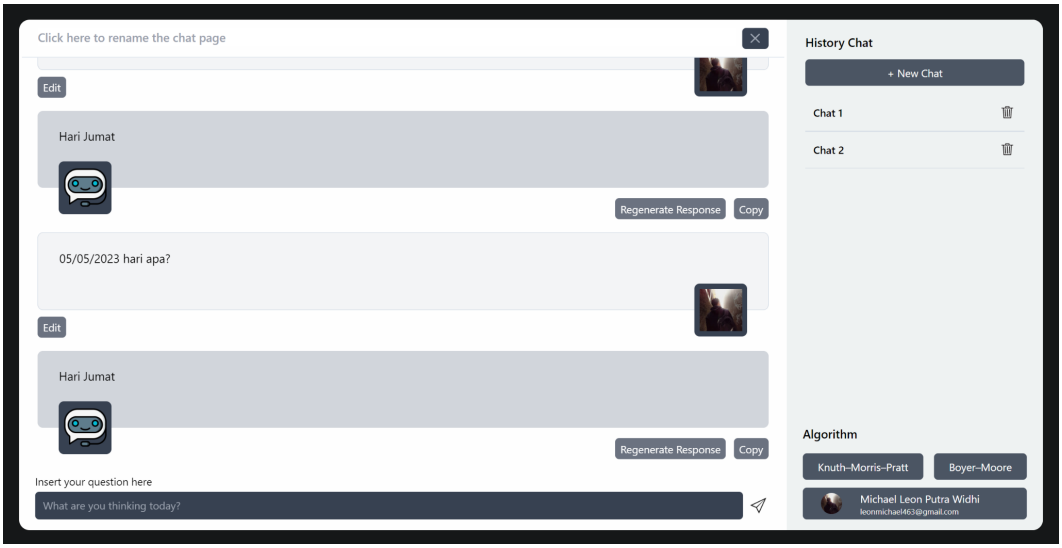
1. Fitur Tanggal



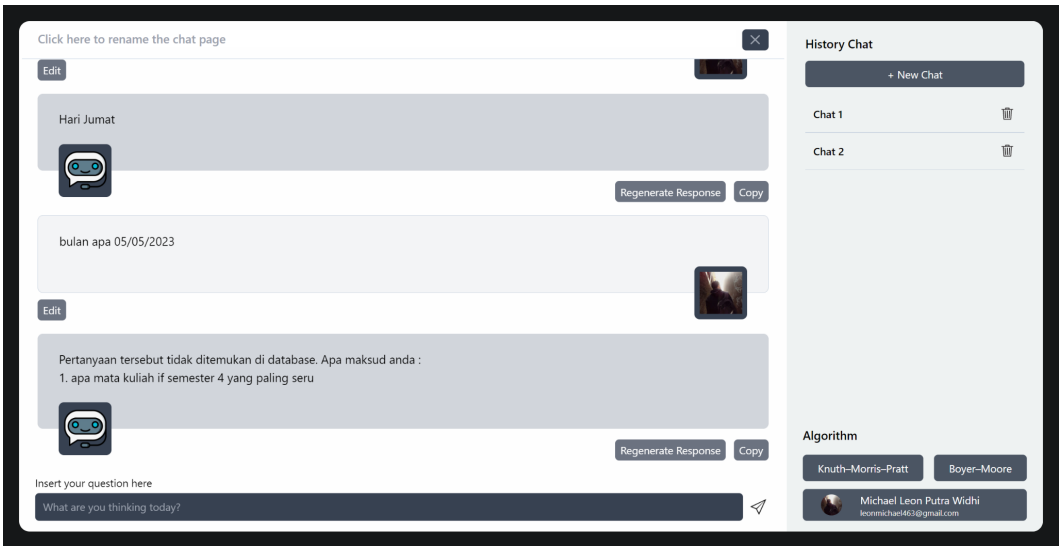
Gambar 4.8.1. Tanggal *valid*.



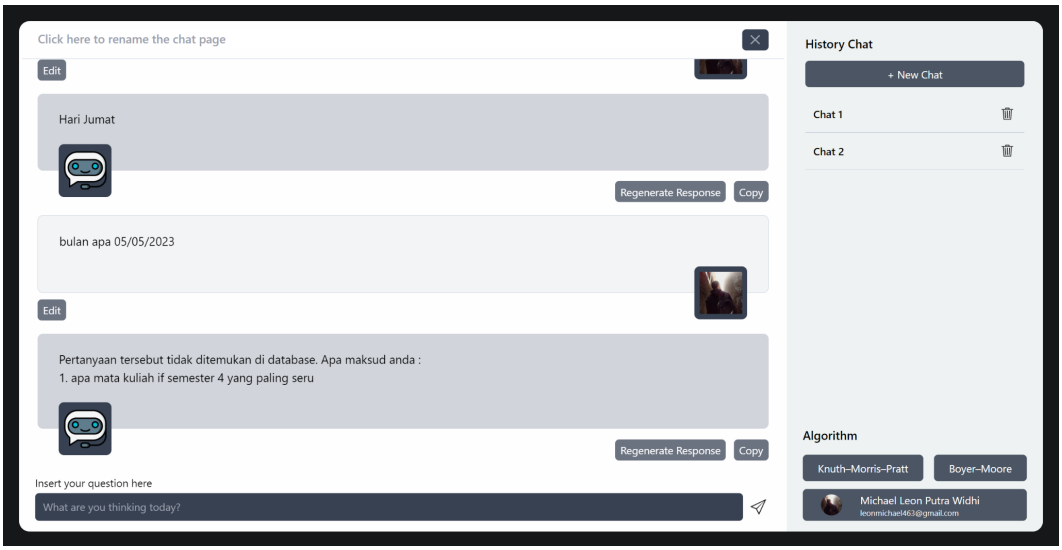
Gambar 4.8.2. Tanggal *valid* dilengkapi kata hari (1).



Gambar 4.8.3. Tanggal *valid* dilengkapi kata hari (2).



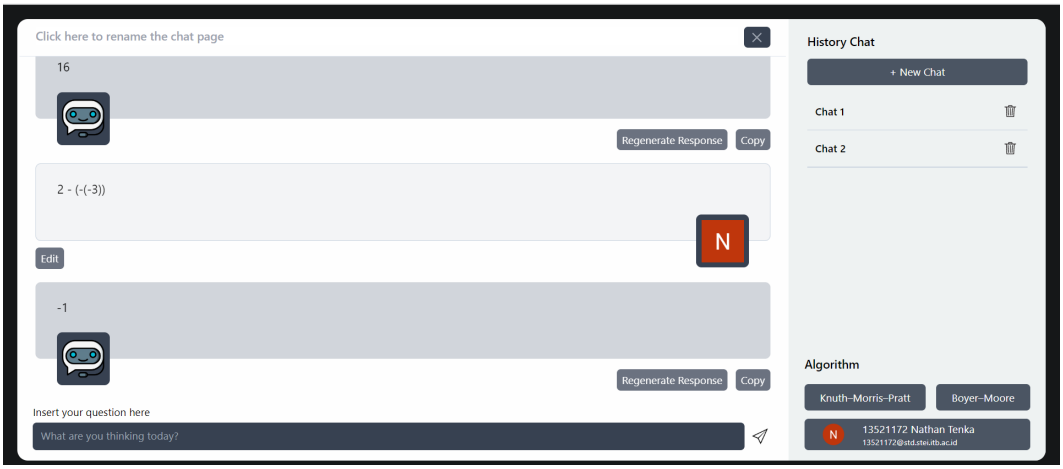
Gambar 4.8.4. Tanggal tidak *valid* dilengkapi bulan.



Gambar 4.8.5. Tanggal tidak *valid* dilengkapi bulan.

2. Fitur Kalkulator

Gambar 4.9.1. Kasus operasi normal.



Gambar 4.9.2. Kasus operasi normal banyak kurung.

Gambar 4.9.3. Kasus operasi perpangkatan desimal.

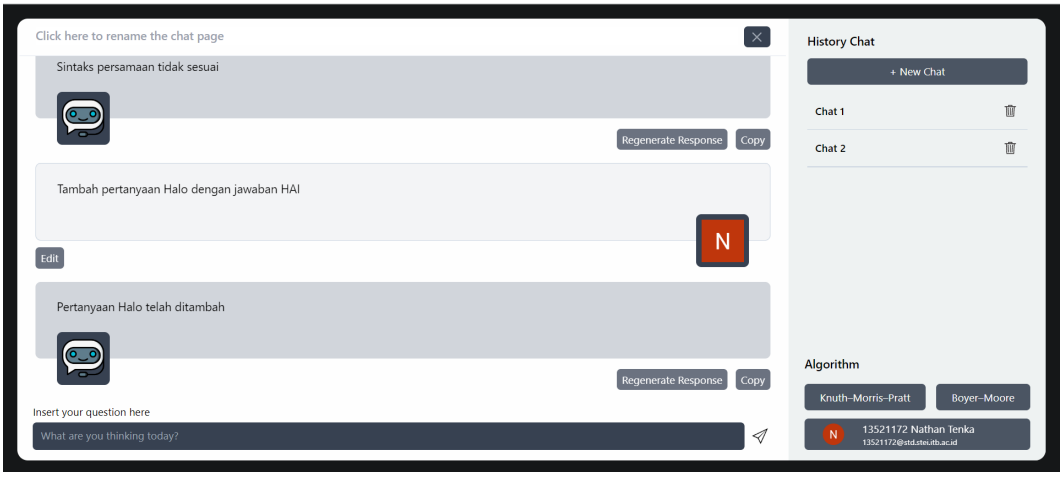
Gambar 4.9.4. Bilangan tidak bulat.

Gambar 4.9.5. Jumlah operator berlebih.

Gambar 4.9.6. Kurung tidak sesuai

Gambar 4.9.7. Ada karakter non-angka dan non-operator di persamaan

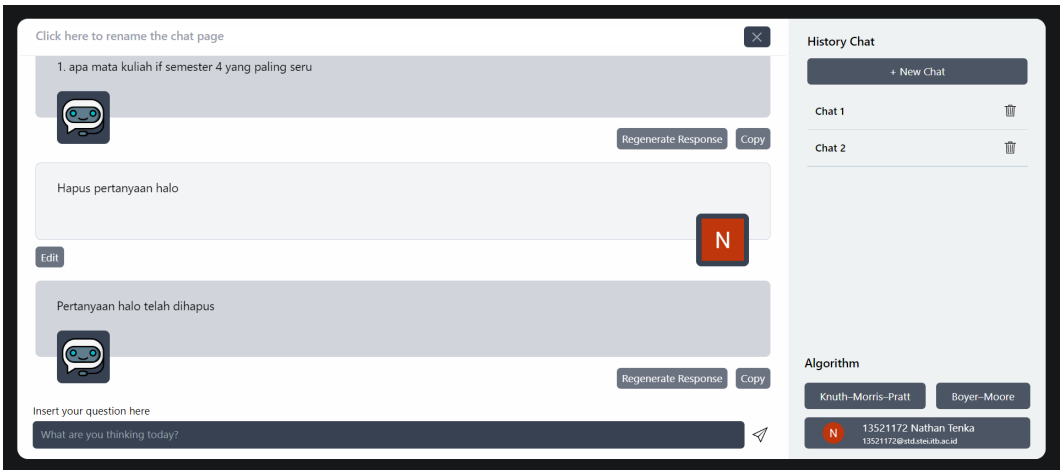
3. Fitur Tambah Pertanyaan



Gambar 4.10.1. Menambah pertanyaan yang belum ada

Gambar 4.10.2. Menambah pertanyaan yang sudah ada

4. Fitur Hapus Pertanyaan



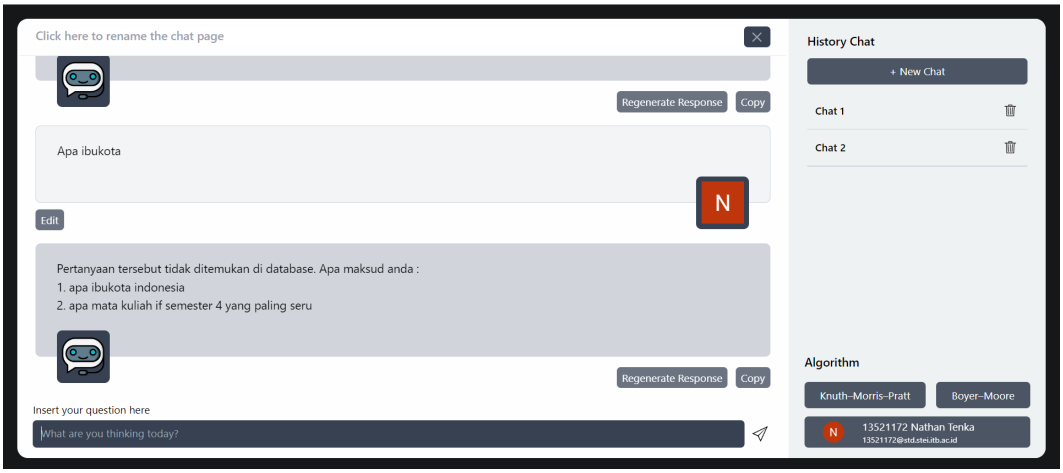
Gambar 4.11.1. Menghapus pertanyaan yang sudah ada

Gambar 4.11.2. Menghapus pertanyaan yang tidak ada

5. Fitur Cari Jawaban

Gambar 4.12.1. Pertanyaan *exact match*

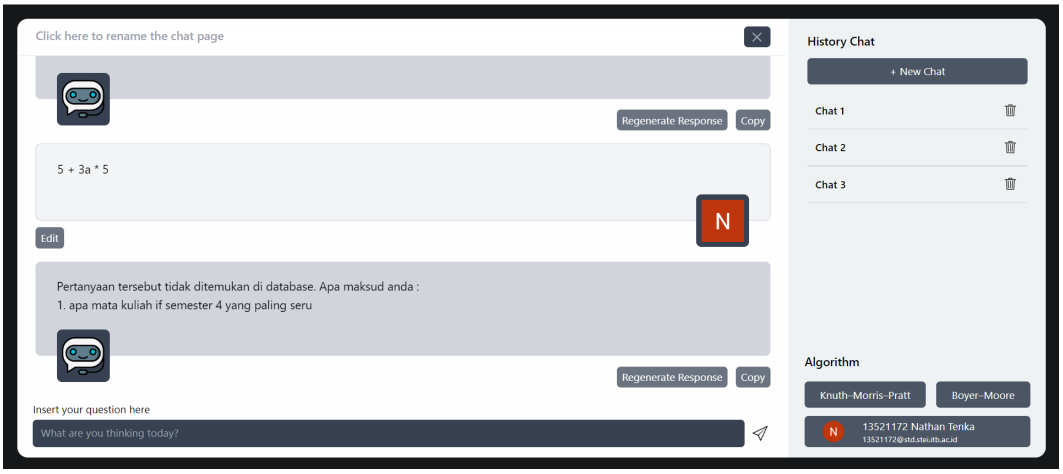
Gambar 4.12.2. Kemiripan ≥ 90%



Gambar 4.12.3. Kemiripan < 90%

6. Fitur *History*

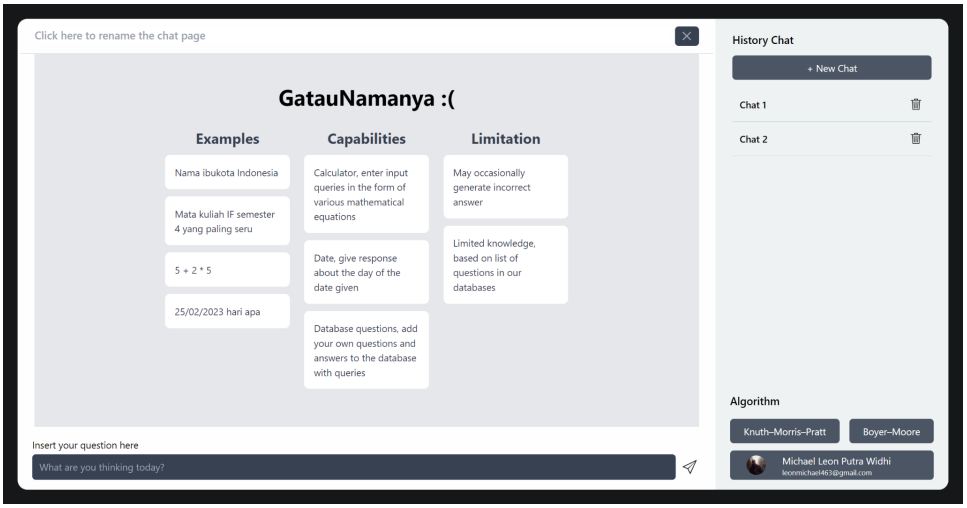
Gambar 4.13.1. Kondisi sebelum *reload*



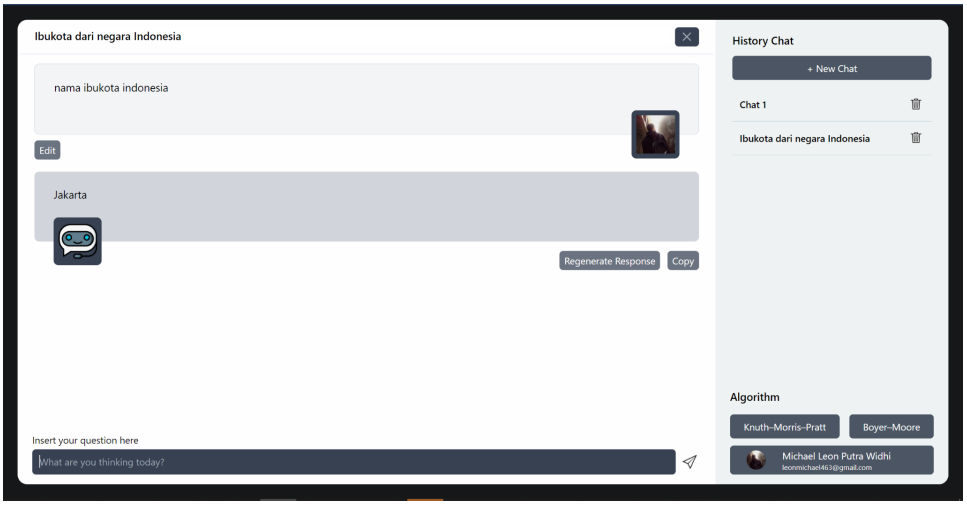
Gambar 4.13.1. Kondisi setelah *reload*

7. Fitur Lain-Lain

7.1. *Rename chat*

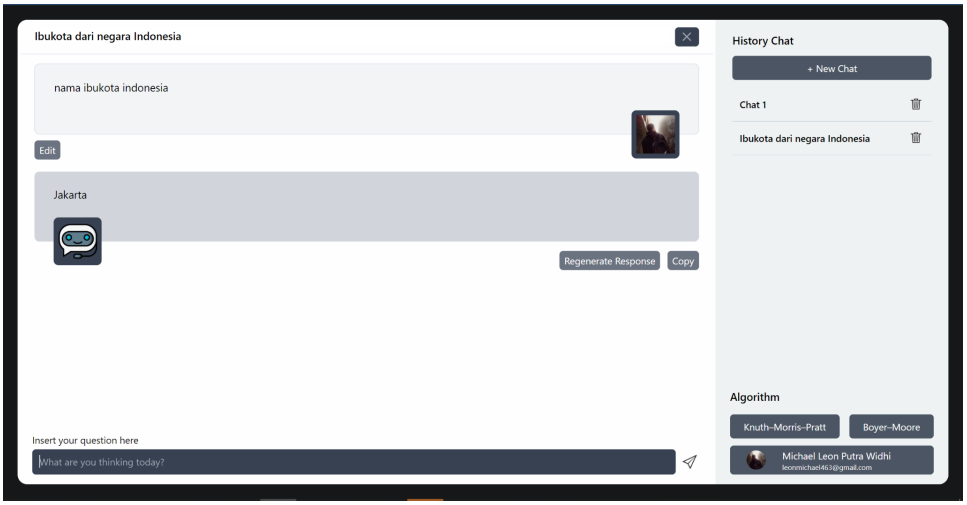


Gambar 4.14.1. Kondisi sebelum penambahan nama *chat*

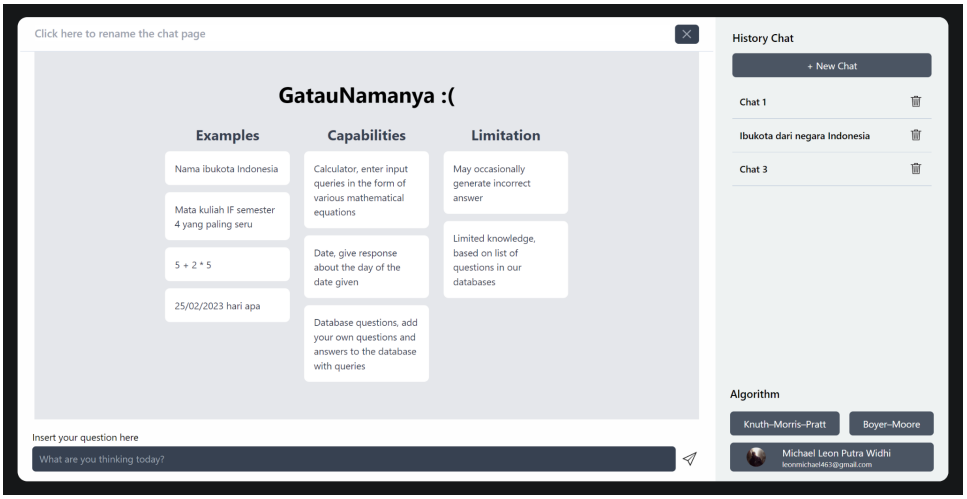


Gambar 4.14.2. Kondisi setelah penambahan nama *chat* dan pengiriman pesan.

7.2. Tambah *chat* baru

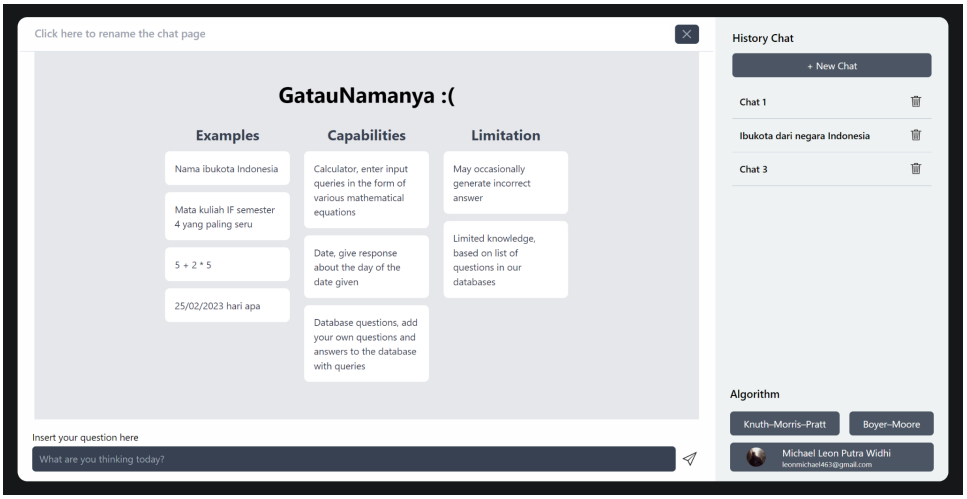


Gambar 4.14.3. Kondisi sebelum penambahan halaman *chat* baru.

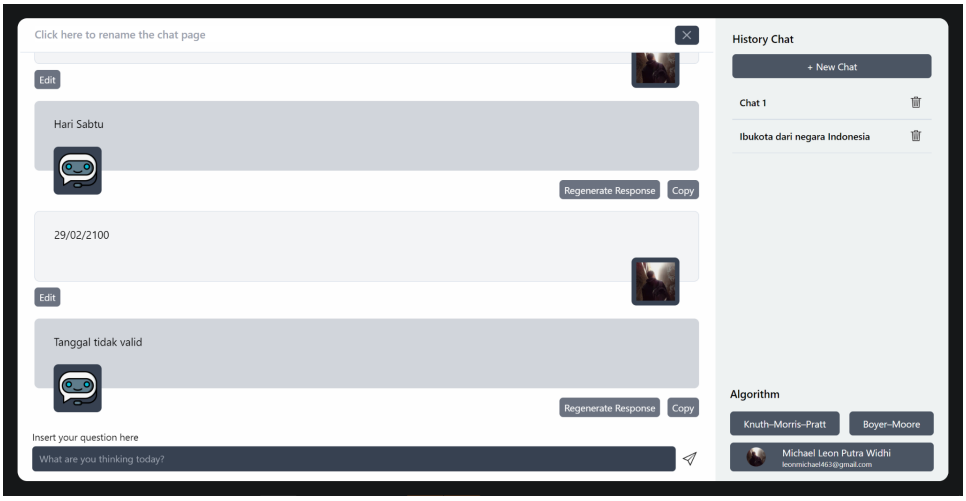


Gambar 4.14.3. Kondisi setelah penambahan halaman *chat* baru.

7.3. Menghapus *chat* yang sudah ada



Gambar 4.14.4. Kondisi sebelum halaman Chat 3 dihapus.



Gambar 4.14.5. Kondisi setelah halaman Chat 3 dihapus.

D. Analisis Hasil Pengujian

1. **Fitur Tanggal**

   Terlihat bahwa sistem bisa memberikan hari dari tanggal yang diberikan dan bisa menolak tanggal yang tidak valid. Sintaks pertanyaan harus tepat (hari (apa)) DD/MM/YYYY (hari (apa) (?)).

   Keterangan : kata/karakter di dalam kurung berarti opsional.

2. **Fitur Kalkulator**

   Terlihat bahwa fitur kalkulator sudah bisa menangani kasus persamaan yang benar maupun tidak valid. Kalkulator juga bisa menangani bilangan negatif dan bilangan tidak bulat.

3. **Fitur Tambah Pertanyaan**

   Terlihat bahwa sistem sudah bisa menambah pertanyaan pada *database* dan bisa meng-*update* jawaban dari pertanyaan bila sudah ada.

4. **Fitur Hapus Pertanyaan**

   Terlihat bahwa sistem bisa menghapus pertanyaan dari *database* dan menolak menghapus pertanyaan yang belum ada di *database*.

5. **Fitur Cari Jawaban**

   Terlihat bahwa sistem bisa mencari jawaban untuk pertanyaan yang sama persis atau mirip $\geq$ 90% dengan pertanyaan yang ada di *database*. Untuk pertanyaan yang kemiripannya $<$ 90%, sistem menyatakan bahwa pertanyaan tidak ditemukan dan

memberi tebakan pertanyaan yang masih memiliki kemiripan dengan pertanyaan yang diberikan.

6. **Fitur *History***

   Terlihat bahwa sistem bisa menyimpan pertanyaan dan jawaban dari sesi sebelumnya dan memunculkannya kembali di sesi terbaru.

7. **Fitur Lain-Lain**

   7.1. *Rename chat*

   Terlihat bahwa sistem bisa memberikan nama pada halaman sehingga halaman kedua memiliki nama sesuai dengan nama yang diinginkan.

   7.2. Tambah *chat* baru

   Terbentuk sebuah halaman *chat* yang baru pada saat tombol "+ New Chat" ditekan" sesuai dengan yang dikehendaki.

   7.3. Hapus *chat* yang sudah ada

   Halaman *chat* yang lama akan terhapus pada saat tombol bergambar tempat sampah di samping halaman yang dikehendaki untuk dihapus, ditekan.

BAB V
KESIMPULAN DAN SARAN

A. Kesimpulan

Algoritma Knuth–Morris–Pratt dan algoritma Boyer-Moore merupakan dua buah algoritma pencocokan *string* yang banyak digunakan untuk memecahkan berbagai masalah yang berkaitan dengan pencarian informasi dari sebuah paragraf atau kumpulan kalimat dalam jumlah yang masif. Dalam tugas besar ini, kedua algoritma tersebut dimanfaatkan untuk mencari solusi terhadap proses penyelesaian *query* masukan dalam mekanisme *Question-Answering* (QA) yang dimiliki oleh sistem aplikasi berbasis web yang dibangun. Algoritma Knuth-Morris-Pratt melakukan penyamaan pola dari kiri ke kanan, sedangkan algoritma Boyer-Moore melakukan penyamaan pola dari belakang. Pada dasarnya desain dari kedua algoritma ini ditujukan untuk melakukan pencocokan *string* dengan lebih mangkus dan sangkil daripada melakukannnya dengan *brute force*, lebih lanjut, keduanya tentu saja memiliki keunggulan dan kelemahannya masing-masing dalam melakukan pemrosesan terhadap larik kata yang panjang.

Proses pembangunan sebuah aplikasi berbasis *web* menjadi bagian yang membawa tantangan tersendiri pada tugas besar kali ini. Proses pengembangan dari kedua belah pihak, baik melalui pihak *frontend* dan *backend* dilakukan dengan menggunakan berbagai kakas yang telah dijabarkan sebelumnya. Proses *deployment* pada aplikasi membuat aplikasi dapat diakses secara global dan diharapkan dapat berguna bagi banyak pihak yang menggunakannya.

B. Saran

Program aplikasi berbasis *web* yang dibangun telah dapat berjalan dengan baik, tetapi program yang telah dibangun masih kurang dari sempurna karena adanya berbagai batasan pengembangan sehingga memungkinkan untuk dikembangkan lebih lanjut, salah satunya dengan menambah fitur-fitur baru dan menggunakan *database* yang lebih besar kapasitasnya. Aplikasi juga bisa dikembangkan dengan menggunakan kecerdasan buatan (*artificial*

intelligence) untuk mendapatkan jawaban yang lebih riil dan variatif dari segi pemrosesan *query* sehingga lebih akurat dengan cakupan jawaban lebih luas.

C. Refleksi

Proses pengimplementasian program, secara umum, berjalan dengan lancar. Walaupun sempat terkendala dengan jumlah waktu yang relatif minimum, tetapi dengan koordinasi yang baik dan menyempatkan diri untuk melakukan pertemuan baik secara daring maupun luring, tugas besar ini dapat selesai dengan cukup baik. Beberapa poin yang menjadi refleksi dari tugas besar ini adalah :

1. Pembagian tugas yang jelas sejak awal pertemuan membuat proses implementasi menjadi lebih terstruktur.
2. Penggunaan komentar (*comment*) sangat membantu dalam memahami program dan langkah dari setiap kode yang diimplementasikan.
3. Pentingnya komunikasi yang baik antar anggota. Ini menjadi aspek yang tidak kalah penting dan menjadi salah satu kunci proses implementasi yang berlangsung dengan kendala yang minimal.
4. Pentingnya melakukan rekam cadang (*back-up*) bagi segmen kode yang memiliki signifikansi besar pada keseluruhan program.
5. Proses pengembangan aplikasi dengan cara *branching* sangat membantu dalam menyelesaikan program dengan lebih efektif.

BAB VI
DAFTAR PUSTAKA

Munir, R. (2023). Pencocokan *String* (*String/Pattern Matching*). IF2211 Strategi Algoritma - Semester II Tahun 2022/2023. Diakses pada 3 Mei 2023, pukul 20.40 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Khodra, Masayu Leylia. (2019). *String Matching* dengan *Regular Expression* (String/Pattern Matching). IF2211 Strategi Algoritma - Semester II Tahun 2022/2023. Diakses pada 3 Mei 2023, pukul 20.45 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf

Geeks For Geeks. (2023). *Longest Common Subsequence* (LCS). Geeks For Geeks. Diakses pada 3 Mei 2023, pukul 20.55 dari https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/

LAMPIRAN

Tautan *Repository* :

https://github.com/mikeleo03/Tubes3_GatauNamanya

Tautan *deployment* :

https://johaneslee.me/

Tautan Video :

https://youtu.be/mPttZ81j9Wk