LAPORAN TUGAS BESAR III

IF2211 STRATEGI ALGORITMA

PENERAPAN STRING MATCHING DAN REGULAR EXPRESSION DALAM PEMBUATAN CHATGPT SEDERHANA

Disusun oleh:

13521062	Go Dillon Audris
13521070	Akmal Mahardika Nurwahyu Pratama
13521071	Margaretha Olivia Haryono

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

Daftar Isi **1**
BAB I **3**
DESKRIPSI TUGAS **3**
1.1 Deskripsi Tugas 3
1.2 Fitur-fitur Aplikasi 4
BAB II **8**
LANDASAN TEORI **8**
2.1 Algoritma Knuth-Morris-Pratt **8**
2.2 Algoritma Boyer-Moore **9**
2.3 Regular Expression **11**
2.4 Levenshtein Distance 12
2.5 Aplikasi Web Yang Dibangun 13
BAB III **15**
ANALISIS PEMECAHAN MASALAH **15**
3.1 Penyelesaian Masalah Tiap Fitur **15**
3.1.1 Klasifikasi Fitur 15
3.1.2 Fitur Pertanyaan (Akses Database) 16
3.1.3 Fitur Kalkulator 16
3.1.4 Fitur Tanggal 17
3.1.5 Fitur Menambahkan Pertanyaan dan Jawaban 17
3.1.6 Fitur Menghapus Pertanyaan 18
3.2 Fitur Fungsional dan Arsitektur Aplikasi **18**
3.2.1 Fitur Fungsional 18
3.2.2 Arsitektur Aplikasi 19
3.2.2.1 Frontend 19
3.2.2.2 Backend 19
BAB IV **21**
IMPLEMENTASI DAN PENGUJIAN **21**
4.1 Spesifikasi Teknis Program 21
4.1.1 File regex.js 21
4.1.2 File processor.js: 21
4.1.3 File kmp.js 23
4.1.4 File bm.js 23
4.1.5 File levenshtein-distance.js 24
4.1.6 File add.js 24
4.1.7 File delete.js 24

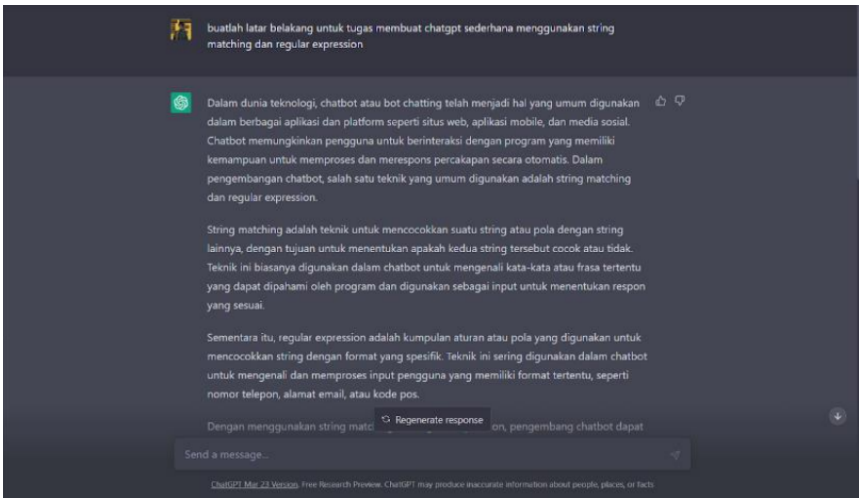
4.1.8 File find.js 24
4.1.9 File calculator.js 25
4.1.10 File date.js 26
4.2 Tata Cara Penggunaan Program 26
4.3 Hasil Pengujian 28
4.3.1 Fitur Kalkulator 28
4.3.2 Fitur Tanggal 29
4.3.3 Fitur Menambahkan Pertanyaan dan Jawaban 30
4.3.4 Fitur Pertanyaan 31
4.3.5 Fitur Menghapus Pertanyaan 33
4.4 Analisis Hasil Pengujian 34
BAB V **36**
KESIMPULAN DAN SARAN **36**
5.1 Kesimpulan 36
5.2 Saran 36
5.3 Komentar dan Refleksi 36
DAFTAR PUSTAKA **38**
Lampiran **39**
Link Repository 40
Link Deployment Web 40
Link Youtube 40

BAB I

DESKRIPSI TUGAS

1.1 Deskripsi Tugas

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 *string Knuth-Morris-Pratt* (KMP) dan *Boyer-Moore* (BM). *Regular expression* 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*.



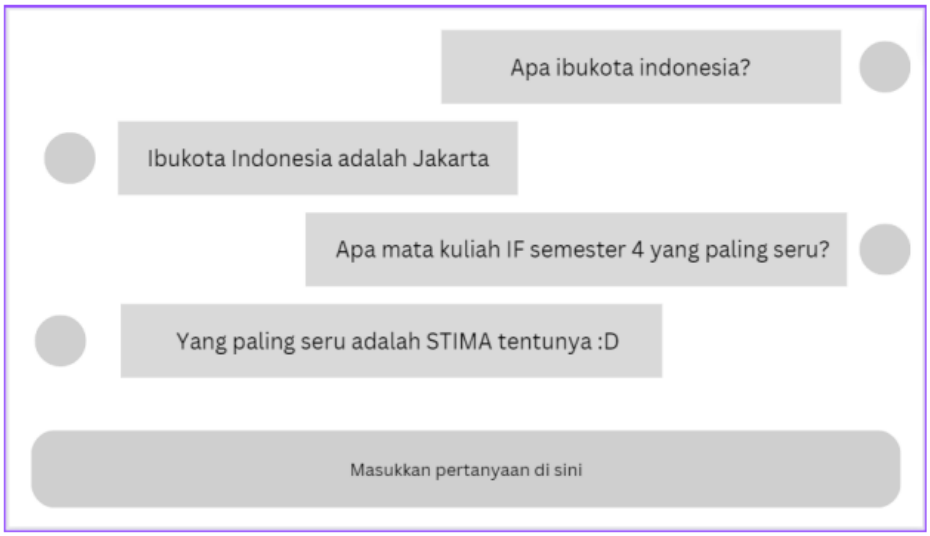
Gambar 1.1.1 Ilustrasi *Chatbot ChatGPT*

1.2 Fitur-fitur Aplikasi

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

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

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

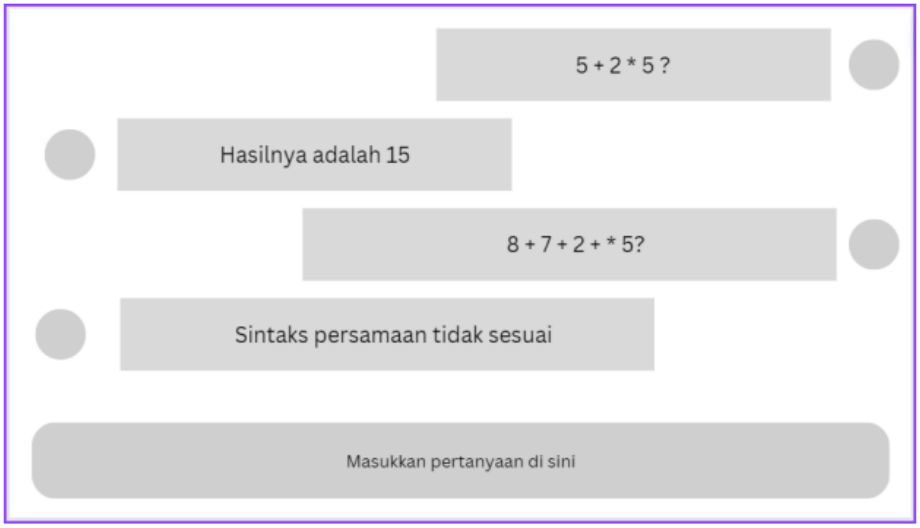


Gambar 1.2.1 Ilustrasi fitur pertanyaan teks kasus *exact*

Gambar 1.2.2 Ilustrasi fitur pertanyaan teks kasus tidak *exact*

2. **Fitur kalkulator :**

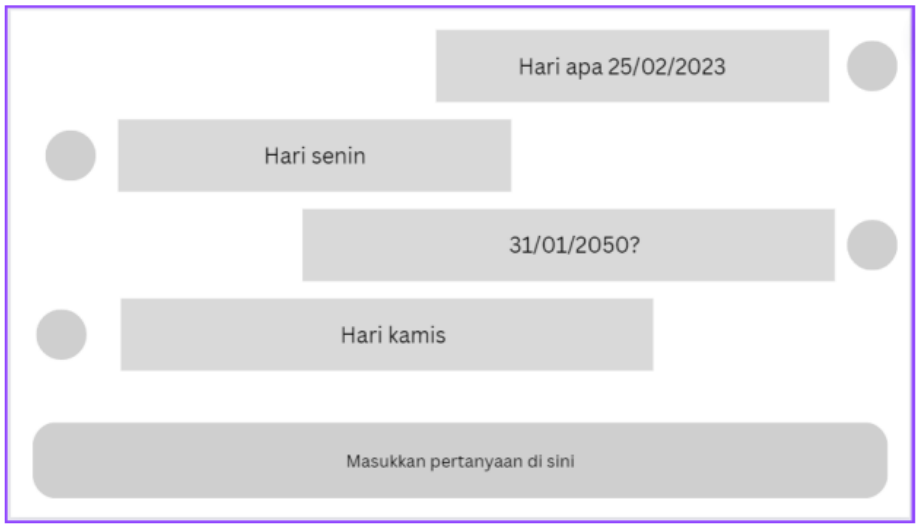
Pengguna memasukkan input query berupa persamaan matematika. Contohnya adalah 2*5 atau 5+9*(2+4). Operasi cukup tambah, kurang, kali, bagi, pangkat, kurung.



Gambar 1.2.3 Ilustrasi fitur kalkulator

3. **Fitur tanggal :**

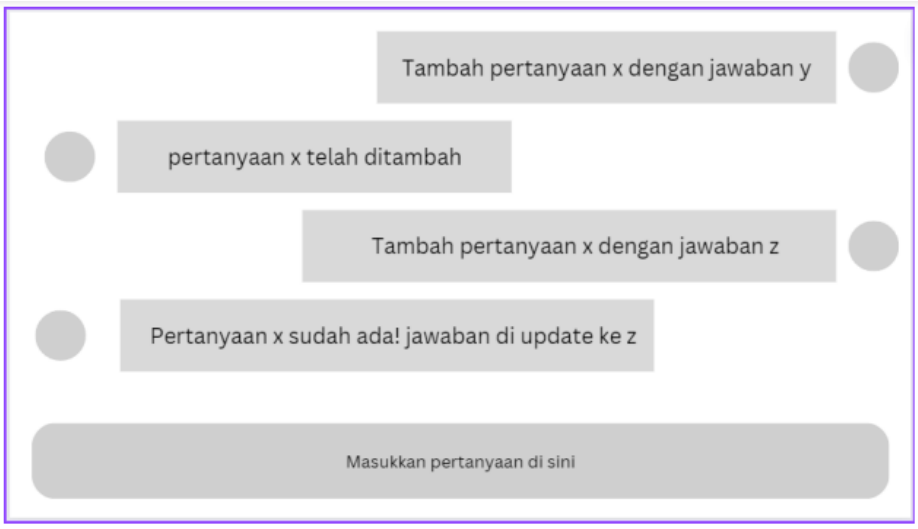
Pengguna memasukkan input berupa tanggal, lalu *chatbot* akan merespon dengan hari apa di tanggal tersebut. Contohnya adalah 25/08/2023 maka *chatbot* akan menjawab dengan hari senin.



Gambar 1.2.4 Ilustrasi fitur tanggal

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

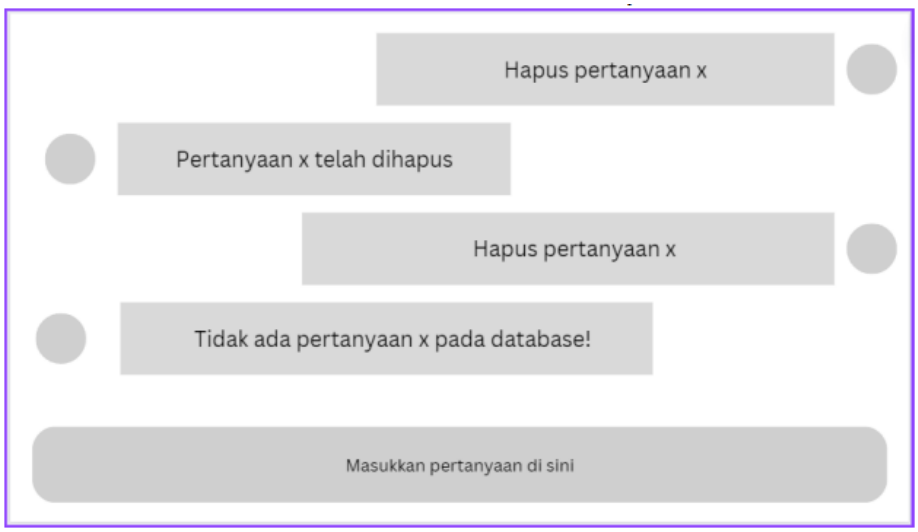
Pengguna dapat menambahkan pertanyaan dan jawabannya sendiri ke *database* dengan *query* contoh "Tambahkan pertanyan xxx dengan jawaban yyy". Menggunakan algoritma *string matching* untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbaharui.



Gambar 1.2.5 Ilustrasi fitur tambah pertanyaan

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

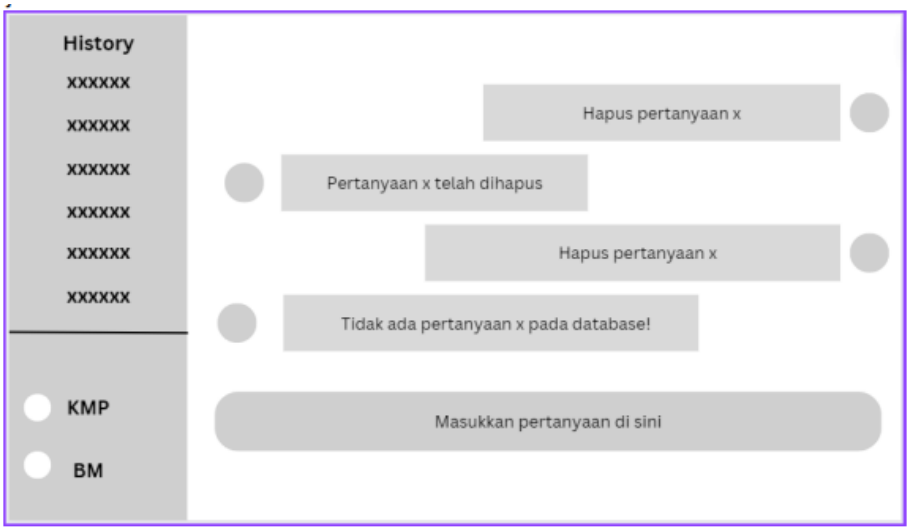
Pengguna dapat menghapus sebuah pertanyaan dari *database* dengan *query* contoh "Hapus pertanyaan xxx". Menggunakan string algoritma *string matching* untuk mencari pertanyaan xxx tersebut pada *database*. Klasifikasi dilakukan menggunakan *regular expression* dan terklasifikasi layaknya bahasa sehari - 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.



Gambar 1.2.6 Ilustrasi fitur hapus pertanyaan

6. ***History*** **:**

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 ditekan menyimpan seluruh pertanyaan pada sesi itu. Apabila *history* ditekan, maka akan me-*restore* seluruh pertanyaan dan jawaban di halaman utama.



Gambar 1.2.7 Ilustrasi keseluruhan

BAB II
LANDASAN TEORI

2.1 Algoritma *Knuth-Morris-Pratt*

Algoritma *Knuth-Morris-Pratt* merupakan sebuah algoritma pencocokan *string* yang mirip dengan algoritma *brute-force*, dimana algoritma ini akan suatu pola pada teks secara berurut dari kiri ke kanan. Namun, algoritma *Knuth-Morris-Pratt* melakukan pergeseran pola secara lebih cerdas dibandingkan dengan algoritma *brute-force*.

Sebelum menjelaskan bagaimana cara algoritma *Knuth-Morris-Pratt* menggeser pola, perlu diketahui terlebih dahulu mengenai *prefix* (awalan) dan *suffix* (akhiran), dimana *prefix* menunjukkan setiap bagian kata yang berada di awal, dan *suffix* menunjukkan setiap bagian kata yang berada di akhir. Jika terjadi sebuah *mismatch* atau ketidaksamaan dalam pencocokan string pada karakter ke-j dalam pola, maka algoritma *Knuth-Morris-Pratt* akan menggeser pola sebanyak ukuran terbesar *prefix* bagian kata ke-0 sampai j-1 yang juga merupakan *suffix* dari bagian kata ke-1 sampai j-1. Pergeseran yang dilakukan ini akan mengurangi komparasi atau perbandingan yang tidak diperlukan sehingga menyebabkan algoritma *Knuth-Morris-Pratt* lebih cepat dibandingkan dengan algoritma *brute-force.*

Akibat cara algoritma KMP bekerja, maka dibutuhkan suatu fungsi pinggiran (*border function*) yang mendefinisikan seberapa banyak pola harus digeser jika terjadi kesalahan pada tiap karakter di dalam pola. Berikut merupakan contoh kode algoritma KMP dalam bahasa *Java*:

```
public static int[] computeBorder(String pattern) {
    int b[] = new int[pattern.length()];
    b[0] = 0;
    int m = pattern.length();
    int j = 0;
    int i = 1;
    while (i < m) {
        if (pattern.charAt(j) == pattern.charAt(i)) {
            b[i] = j + 1;
            i++;
            j++;
        }
        else if (j > 0)  {
```

```
            j = b[j-1];
        }
        else {
            b[i] = 0;
            i++;
        }
    }
    return b;
}

public static int kmpMatch(String text, String pattern) {
    int n = text.length();
    int m = pattern.length();
    int b[] = computeBorder(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;
            i++;
            j++;
        }
        else if (j > 0)
            j = b[j-1];
        else
            i++;
    }
    return -1;
}
```

2.2 Algoritma *Boyer-Moore*

Algoritma *Boyer-Moore* merupakan sebuah algoritma pencocokan *string* yang efisien untuk mencari sebuah pola dalam sebuah teks. Algoritma ini bekerja dengan cara memanfaatkan informasi mengenai karakter yang tidak cocok untuk menggeser pola lebih jauh, sehingga dapat mengurangi jumlah perbandingan atau komparasi yang dibutuhkan.

Terdapat beberapa pendekatan algoritma *Boyer-Moore* yaitu Pendekatan *Good Suffix*, *Bad Character* dan gabungan keduanya. Ketika terjadi ketidak cocokan karakter pada pendekatan *Good Suffix*, *string* digeser berdasarkan pola *sufix* yang terakhir. Sedangkan, pendekatan *Bad Character* melakukan pergeseran *string* berdasarkan

karakter terakhir muncul di string yang dicocokan. Pada tugas besar kali ini, Pendekatan *Bad Character* digunakan.

Algoritma Boyer-Moore pendekatan *Bad Character* melakukan dua teknik dalam pencocokan *string*. Dua teknik tersebut adalah looking-glass dan character-jump. Teknik looking-glass dilakukan dengan mencari pola secara mundur, dimulai dari akhir pola untuk menemukan kemungkinan kesamaan dengan teks. Sedangkan, teknik character-jump digunakan saat terjadi ketidakcocokan karakter dalam pencarian, dan dilakukan dengan memanfaatkan informasi mengenai posisi terakhir kemunculan karakter yang tidak cocok pada pola, untuk menggeser pola ke kanan sejauh mungkin. Dalam implementasinya, teknik character-jump dapat menggunakan sebuah tabel yang disebut jumping table, yang berisi informasi mengenai jumlah karakter yang harus digeser pada saat terjadi ketidakcocokan karakter.

Ketika sebuah karakter tidak cocok, algoritma akan memeriksa *character-jump*. Pergeseran dilakukan berdasarkan kasus yang terjadi. Terdapat tiga kasus dalam algoritma ini, yaitu kasus 1 jika pola P mengandung karakter x, maka akan digeser ke kanan agar karakter terakhir x pada P sejajar dengan T[i]. Kasus 2 terjadi jika P juga mengandung x, tetapi tidak memungkinkan untuk menggeser pola hingga karakter x terakhir pada P sejajar dengan T[i], maka P akan digeser ke kanan sebanyak 1 karakter pada T[i+1]. Sedangkan pada kasus 3, jika kedua kasus sebelumnya tidak terpenuhi, maka pola P akan digeser hingga P[0] sejajar dengan T[i+1]. Dengan pendekatan ini, algoritma Boyer-Moore mampu mengurangi jumlah perbandingan atau komparasi yang dibutuhkan dalam pencocokan string.

Oleh karena itu, fungsi lastOccurBuilder dibutuhkan untuk melakukan dan membentuk *character-jump*. Berikut merupakan contoh kode algoritma dalam bahasa javascript.

```
function lastOccurBuilder(pattern) {
    var last = new Array(256);
    for(let i=0; i < 256; i++) {
        last[i] = -1
    }
    for (let i = 0; i < pattern.length; i++) {
        last[pattern.charCodeAt(i)] = i
    }
```

```
    return last
}

function matchBM(text, pattern) {
    text = text.toLowerCase()
    pattern = pattern.toLowerCase()

    var lastOccur = lastOccurBuilder(pattern)
    var lengthText = text.length
    var lengthPattern = pattern.length
    var i = lengthPattern - 1

    if (i > lengthText - 1) {
        return -1
    }

    var j = lengthPattern - 1

    do {
        if (pattern[j] == text[i]) {
            if (j == 0) {
                // match
                return i
            }
            else {
                i--
                j--
            }
        }
        else {
            var lo = lastOccur[text[i]]
            i = i + lengthPattern - Math.min(j, 1 + lo)
            j = lengthPattern - 1
        }
    } while (i <= lengthText - 1)
    return -1
}
```

2.3 *Regular Expression*

Regular expression (*regex*) adalah suatu bahasa formal yang banyak digunakan untuk melakukan pencocokan, validasi, maupun hal lain terhadap *string*. Teori *regex* berasal dari konsep teori bahasa formal dan automata. *Regex* didefinisikan sebagai sebuah kumpulan simbol atau karakter terbatas yang membentuk suatu pola tertentu. Pola inilah yang kemudian akan dicocokan pada suatu teks. *Regex* juga dimanfaatkan sebagai input dalam automata deterministik dan non-deterministik (DFA dan NDFA).

Dalam implementasinya, *regex* memiliki sintaks serta karakter khusus yang memungkinkan terbentuknya suatu pola yang digunakan untuk melakukan pencocokan string. Berikut adalah daftar karakter *regex* secara umum yang banyak digunakan pada berbagai bahasa pemrograman:

Symbol	Description	Symbol	Description
^	Start of line +	?	0 or 1 +
\A	Start of string +	{3}	Exactly 3 +
$	End of line +	{3,}	3 or more +
\Z	End of string +	{3,5}	3, 4 or 5 +
\b	Word boundary +	\	Escape Character +
\B	Not word boundary +	\n	New line +
\<	Start of word	\r	Carriage return +
\>	End of word	\t	Tab +
\s	White space	.	Any character except new line (\n) +
\S	Not white space	(a|b)	a or b +
\d	Digit	[abc]	Range (a or b or c) +
\D	Not digit	[^abc]	Not a or b or c +
\w	Word	[0-7]	Digit between 0 and 7 +
\W	Not word	[a-q]	Letter between a and q +
*	0 or more +	[A-Q]	Upper case letter + between A and Q +
+	1 or more +		

Gambar 2.3.1 Daftar sintaks *regular expression*

2.4 *Levenshtein Distance*

Levenshtein Distance atau juga dikenal sebagai Edit Distance, adalah sebuah metode pengukuran kesamaan antara dua buah string yang didefinisikan sebagai jumlah minimum operasi edit yang diperlukan untuk mengubah satu string menjadi string yang lainnya. Operasi edit yang diizinkan dalam Levenshtein Distance adalah insert (penambahan karakter), delete (penghapusan karakter), dan substitute (penggantian karakter).

Dalam aplikasi nyata, Levenshtein Distance dapat digunakan untuk memberikan rekomendasi kata terdekat ketika pengguna melakukan kesalahan ketik, mendeteksi kemiripan teks antara dua dokumen untuk menemukan kemungkinan plagiarisme, atau untuk membandingkan transkripsi suara dengan teks yang seharusnya terucap. Oleh karena itu, Levenshtein Distance merupakan sebuah konsep yang penting dalam bidang

pemrosesan bahasa alami dan dapat memberikan solusi untuk berbagai masalah yang melibatkan perbandingan teks dan pengenalan pola.

Dalam tugas besar ini, fungsi Levenshtein Distance dimodifikasi. Kembalian fungsi tersebut yang asalnya menunjukkan banyak *insert*, *delete*, dan substitusi menjadi mengembalikan perbandingan banyak kesalahan per panjang *string* dengan panjang *string* tersebut adalah panjang maksimal antara string pertanyaan dan string dari basis data. Perbandingan tersebut digunakan untuk menampilkan pertanyaan-pertanyaan pada basis data yang mirip dengan range kecocokan 45% - 90%

2.5 Aplikasi Web Yang Dibangun

Aplikasi web yang kami bangun adalah sebuah aplikasi *chatbot* yang memanfaatkan algoritma *Knuth-Morris-Pratt* dan *Boyer-Moore* dalam menentukan jawaban yang sesuai dari pertanyaan masukan. Aplikasi ini dikembangkan menggunakan bahasa Javascript dengan memanfaatkan MERN *stack* (MongoDB, Express, React, Node). Secara umum, struktur kode program terbagi menjadi dua bagian, yaitu *frontend* untuk *client-side* dan *backend* untuk *server-side*.

Frontend adalah bagian dari suatu aplikasi atau situs web yang terlihat oleh pengguna dan berinteraksi langsung dengan pengguna. Pada aplikasi yang dibangun, kami menggunakan React, yang adalah sebuah pustaka atau *library* JavaScript untuk membangun antarmuka pengguna atau *user interface* pada aplikasi web. Untuk memperindah tampilannya, kami menggunakan pustaka CSS bernama Tailwind. Selain itu, pada bagian *frontend* dibutuhkan pengambilan data yang nantinya akan ditampilkan pada antarmuka web. Oleh karena itu, kami memanfaatkan Axios yang merupakan pustaka JavaScript yang digunakan untuk melakukan HTTP *request* ke server. Sehingga, kami dapat mengambil data-data yang dibutuhkan melalui API dari *backend*.

Backend adalah bagian dari suatu aplikasi yang berada di sisi server dan bertanggung jawab untuk mengelola data dan berinteraksi dengan *database*. Selain itu, pada bagian *backend* terdapat juga algoritma *string matching* yang digunakan untuk memproses pertanyaan masukan pengguna pada bagian *frontend* untuk mendapatkan jawabannya. *Database* yang kami gunakan adalah MongoDB. Beberapa pustaka Javascript yang kami manfaatkan dalam membangun sisi server ini adalah *mongoose*

untuk menghubungkan koneksi ke *database* dan Express untuk membuat *router* sebagai API yang nantinya dipanggil oleh *frontend*.

BAB III
ANALISIS PEMECAHAN MASALAH

3.1 Penyelesaian Masalah Tiap Fitur

3.1.1 Klasifikasi Fitur

Masukan (*input*) pertanyaan dari pengguna aplikasi *web* harus diklasifikasikan menjadi berbagai fitur yang yang disediakan oleh *chatbot*. Klasifikasi ini diperlukan untuk memberikan respons *chatbot* yang benar berkaitan dengan pertanyaan yang diberikan oleh pengguna. Klasifikasi *query* dilakukan dengan memanfaatkan *regular expression* untuk mencocokkan tiap pertanyaan dengan pola tertentu yang dimiliki oleh fitur tertentu.

Sebagai contoh, fitur kalkulator memiliki pola pertanyaan dimana masukan pengguna hanya berisi ekspresi matematika (angka dan operator matematika umum). Sedangkan fitur menambahkan pertanyaan dan jawaban memiliki pola dimana terdapat frasa "tambahkan pertanyaan" dan "dengan jawaban". Pola-pola ini yang kemudian menjadi bagian dari *regular expression* untuk mengklasifikan masukan pertanyaan menjadi fitur yang sesuai.

Secara umum, klasifikasi fitur dilakukan dengan melakukan tes masukan pertanyaan terhadap *regular expression* tersebut dan tes dilakukan secara prioritas. Sebagai contoh, tes terhadap *regular expression* dari fitur tanggal akan dilakukan terlebih dahulu dibandingkan fitur kalkulator karena masukan mengenai fitur tanggal juga hanya mengandung ekspresi matematika yang diterima oleh *regular expression* fitur kalkulator. Contoh lain adalah fitur pertanyaan diletakkan di akhir prioritas, hal ini karena pertanyaan dari pengguna dapat berupa apa saja dan tidak memiliki suatu pola tertentu. Akibatnya, cukup dilakukan pengecekan *regular expression* untuk fitur-fitur lain terlebih dahulu. Ketika tidak terdapat *regular expression* yang sesuai dengan fitur yang lain, maka masukan pengguna akan langsung dianggap sebagai fitur pertanyaan.

3.1.2 Fitur Pertanyaan (Akses *Database*)

Fitur pertanyaan merupakan fitur aplikasi *chatbot* dimana pengguna dapat memberikan suatu pertanyaan kepada *chatbot*. *Chatbot* kemudian akan melakukan akses *database* untuk mencari pertanyaan yang serupa dengan masukan pengguna. Jika ditemukan pertanyaan tersebut, maka *chatbot* akan memberikan respon berupa jawaban dari pertanyaan yang disimpan di *database*.

Untuk mengimplementasikan fitur ini, maka aplikasi perlu melakukan akses ke *database* untuk mengambil semua *record* yang disimpan. Terhadap setiap atribut pertanyaan yang ada, maka *chatbot* akan melakukan algoritma pencocokan *string* (*Knuth-Morris-Pratt* atau *Boyer-Moore*, tergantung pilihan pengguna) untuk menemukan pertanyaan yang *exact-match*. Jika pertanyaan tersebut ditemukan, maka *chatbot* akan memberikan respons berupa atribut jawaban dari *record* tersebut. Namun, jika *record* yang *exact-match* belum ditemukan, *chatbot* akan memproses kembali setiap atribut pertanyaan dan mencari pertanyaan dengan tingkat kemiripan sebesar 90% atau lebih. Algoritma kemiripan dilakukan dengan memanfaatkan *levenshtein distance* antara pertanyaan di *database* dengan masukan pengguna.

Jika atribut pertanyaan yang memenuhi hal-hal yang diatas, maka *chatbot* akan memberikan beberapa rekomendasi pertanyaan yang mungkin dimaksud oleh pengguna. Pertanyaan tersebut adalah pertanyaan-pertanyaan dengan tingkat kemiripan diatas 45% dengan masukan pengguna. Jika bahkan tidak ada pertanyaan dengan tingkat kemiripan di atas 45%, maka *chatbot* akan memberikan pesan bahwa masukan pengguna tidak dapat diproses.

3.1.3 Fitur Kalkulator

Fitur kalkulator merupakan fitur lain aplikasi *chatbot* yang dapat memberikan hasil evaluasi ekspresi matematika yang diberikan oleh pengguna. Untuk memproses ekspresi matematika, maka *chatbot* akan memanfaatkan dua buah *array* atau *list* yang berfungsi layaknya sebuah *stack*. Satu *stack* digunakan untuk menyimpan operan (angka) dan *stack* lain digunakan untuk menyimpan operator. Terdapat suatu fungsi yang akan menentukan *presedensi* dari tiap

operator, dengan urutan perpangkatan, perkalian dan pembagian, serta penambahan dan pengurangan. Fungsi ini akan menentukan bagaimana ekspresi matematika akan dievaluasi.

Di akhir proses evaluasi, *chatbot* akan melihat keadaan akhir *stack*. Jika *stack* operan bersisa 1 angka dan *stack* operator kosong, maka ekspresi matematika merupakan ekspresi yang valid dan hasil akan diberikan kepada pengguna. Namun, jika terjadi kesalahan pada proses evaluasi, maka *chatbot* akan memberikan pesan bahwa ekspresi matematika yang diberikan oleh pengguna tidak valid dan tidak dapat dievaluasi.

3.1.4 Fitur Tanggal

Fitur tanggal yang dimiliki oleh *chatbot* dapat menerima suatu format tanggal dalam format DD/MM/YYYY. Tanggal dan bulan dapat berupa 1 atau 2 angka dan tahun dapat berupa 1 sampai 4 angka. Format tanggal yang diberikan oleh pengguna kemudian akan diproses oleh *chatbot* untuk memberikan nama hari pada tanggal tersebut.

Dalam melakukan pemrosesan tanggal, *chatbot* memanfaatkan *library Date* yang disediakan oleh bahasa *JavaScript*. *Library* ini akan mengembalikan nama hari yang sesuai pada tanggal tersebut. Sebelum nama hari ini diberikan kepada pengguna, terlebih dahulu akan dicek apakah format tanggal yang diminta benar-benar ada (pengecekan rentang tanggal, bulan, tahun, serta tahun kabisat). Jika tanggal valid, maka nama hari akan diberikan, dan jika tidak, maka *chatbot* akan memberikan respon bahwa tanggal tidak valid.

3.1.5 Fitur Menambahkan Pertanyaan dan Jawaban

Fitur menambahkan pertanyaan dan jawaban merupakan fitur lain yang memungkinkan pengguna untuk memasukkan suatu pertanyaan dan jawaban ke *database*. Fitur ini dapat diakses jika pengguna memasukkan input berupa “tambahkan pertanyaan *pertanyaan* dengan jawaban *jawaban*“. Ketika input dengan pola seperti ini dimasukkan, maka *chatbot* akan mengakses *database*

terlebih dahulu untuk menemukan pertanyaan dengan memanfaatkan algoritma KMP dan BM.

Jika pertanyaan tersebut ditemukan, maka atribut jawaban akan diperbarui dan *chatbot* akan memberikan respon bahwa jawaban telah diperbarui. Namun, jika pertanyaan tersebut tidak ditemukan, maka *chatbot* akan menambahkan *record* pertanyaan dan jawaban yang baru serta memberikan respon bahwa pertanyaan baru telah ditambahkan ke *database*.

3.1.6 Fitur Menghapus Pertanyaan

Fitur menghapus pertanyaan memungkinkan pengguna untuk menghapus pertanyaan yang sudah ada di dalam *database*. Fitur ini dapat diakses jika pengguna memasukkan input berupa "hapus pertanyaan *pertanyaan*". Ketika input dengan pola seperti ini dimasukkan, maka *chatbot* akan mengakses *database* terlebih dahulu untuk menemukan pertanyaan dengan memanfaatkan algoritma KMP dan BM.

Jika pertanyaan tersebut ditemukan, maka *record* dengan pertanyaan tersebut akan dihapus dari *database* dan *chatbot* akan memberi respon bahwa *record* telah dihapus. Namun, jika pertanyaan tersebut tidak ditemukan, maka *chatbot* memberikan respon bahwa pertanyaan yang diminta untuk dihapus tidak ditemukan di *database*.

3.2 Fitur Fungsional dan Arsitektur Aplikasi

3.2.1 Fitur Fungsional

Selain fitur-fitur yang telah dijelaskan di atas, fitur fungsional lain pada aplikasi yang dapat diakses oleh pengguna adalah sebagai berikut.

3.2.1.1 Tambah *Tab Chat*

Untuk menambahkan *tab* baru, pengguna dapat menekan tombol *new tab*. Aplikasi akan secara otomatis membuat satu *tab* baru yang kosong.

3.2.1.2 Hapus *Tab Chat*

Pengguna juga dapat menghapus suatu *tab* dengan menekan *icon* tempat sampah pada judul *tab* yang ingin dihapus. Ketika menghapus suatu *tab,*

history chat pada *tab* tersebut secara otomatis akan terhapus. Sehingga, jika pengguna menambahkan *tab* baru, maka *history* sebelumnya akan hilang dan memulai dari *tab* kosong.

3.2.2 Arsitektur Aplikasi

Secara umum, arsitektur aplikasi terbagi menjadi dua bagian sebagai berikut.

3.2.2.1 *Frontend*

Implementasi *frontend* atau *Client-Side* dari aplikasi ini dibuat dengan menggunakan pustaka React JS dengan bantuan pustaka Tailwind CSS untuk memperindah tampilan. Aplikasi yang dibangun menggunakan pendekatan *Single-Page App*, sehingga aplikasi hanya memiliki satu *page* dengan pemanfaatan *react-router-dom* sehingga pengguna dapat berpindah-pindah *room* atau *tab chat*.

Seluruh kode untuk tampilan aplikasi berada pada file *App.js*. Pada tampilan aplikasi, terdapat *side-bar* pada sisi kiri yang berisikan tombol *add tab* beserta daftar seluruh *tab chat.* Terdapat juga *radio button* untuk pilihan algoritma yang digunakan (KMP atau BM). Jika memilih salah satu *tab chat* yang ada, maka akan terdapat komponen *ChatWindow*, yaitu layar untuk menuliskan pertanyaan dan mendapatkan jawaban dari aplikasi. Tampilan layar ini seperti pada aplikasi *chatting* pada umumnya, yaitu terdapat *input text* untuk memasukkan pertanyaan, tombol *send* untuk mengirimkan pertanyaan, serta *history chat* sebelumnya pada *tab* tersebut. Pengambilan data dan hasil dari menjalankan algoritma *string matching* dilakukan dengan menggunakan *axios*.

3.2.2.2 *Backend*

Implementasi *backend* atau *Server-Side* pada aplikasi ini dibuat dengan menggunakan kakas Express.js. Pada *backend*, terdapat folder *models* yang berisi struktur tabel pada *database*. Koneksi antara *database* dengan kode program dilakukan di dalam *file index.js* dengan memanfaatkan pustaka *mongoose*. *Database* yang digunakan untuk

membangun aplikasi ini adalah MongoDB. Berikut daftar tabel yang dibuat pada *database*.

a. history : Menyimpan *history chat* dari pertanyaan yang pernah ditanyakan sebelumnya beserta jawabannya.
b. qna : Menyimpan pertanyaan beserta jawaban yang nantinya digunakan sebagai basis dalam algoritma *string matching*.
c. tab : Menyimpan tab yang ada pada aplikasi.

Pada *backend* terdapat juga folder *routes* yang berisi *endpoint* untuk API yang dibutuhkan oleh *frontend* dalam mengambil, menambahkan, ataupun mengubah data dari *database*. Berikut daftar *route* yang terdapat pada API aplikasi.

a. /chat (GET) : Mengambil semua *history chat*
b. /chat/{id} (GET) : Mengambil *history chat* dengan {id} tab tertentu
c. /chat/{id} (POST) : Menambah *history chat* baru
d. /chat/delete/{id} (DELETE) : Menghapus *history chat* dengan {id} tab tertentu
e. /qna (GET) : Mengambil semua pertanyaan
f. /qna (POST) : Menambah pertanyaan
g. /qna/delete (DELETE) : Menghapus pertanyaan
h. /tab (GET) : Mengambil semua *tab*
i. /tab (POST) : Menambahkan *tab*
j. /tab/delete/{id} (DELETE) : Menghapus *tab*

Kemudian, seluruh algoritma (*string matching*, *regex,* dan perhitungan kemiripan dengan *levenshtein-distance*) serta implementasi fitur terdapat dalam folder *src* pada *backend*. Fungsi *regex* yang digunakan untuk *parsing* pertanyaan masukan dipanggil oleh *router* yang bersesuaian sehingga dapat digunakan oleh *frontend* untuk mendapatkan jawaban atas pertanyaan yang dimasukkan.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Spesifikasi Teknis Program

4.1.1 *File* regex.js

Berisi fungsi-fungsi untuk melakukan pemrosesan awal pada masukan pengguna.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`async function handleQuery(queries, algorithmCode)`	Melakukan pemrosesan awal terhadap masukan pengguna, dimana masukan akan dipisah berdasarkan simbol ";" yang menjadi pemisah antara masukan dengan fitur tertentu dengan masukan lainnya.
2.	`async function processQuery(query, algorithmCode)`	Melakukan pencocokan *string* terhadap parameter *query* dengan memanfaatkan *regular expression*. Fungsi ini kemudian akan memanggil fungsi fitur yang sesuai dengan hasil klasifikasi masukan pengguna.

4.1.2 *File* processor.js:

Berisi fungsi-fungsi untuk melakukan akses dan manipulasi data pada *database*.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`async function findQuestion(question, algorithmCode)`	Mengembalikan ID dari *record* dengan pertanyaan yang sama dengan parameter *question*. Pencocokan *string* dilakukan dengan memanfaatkan algoritma KMP atau BM tergantung dengan parameter *algorithmCode*. Jika pertanyaan yang

		sama tidak ditemukan, maka akan dilakukan algoritma kemiripan dengan memanfaatkan *levenshtein-distance*.
2.	`async function deleteQuestion (question, algorithmCode)`	Menghapus pertanyaan pada *database* yang sesuai dengan parameter *question*. Mengembalikan *string* respon yang sesuai dengan proses penghapusan yang dijalankan.
3.	`async function addQuestion(question, answer, algorithmCode)`	Menambahkan *record* berisi parameter *question* dan *answer* ke dalam *database*. Jika pertanyaan telah ada di dalam *database*, maka akan memperbaharui jawaban dari pertanyaan tersebut. Mengembalikan *string* respon yang sesuai dengan proses penambahan yang terjadi.
4.	`async function findAnswer(question, algorithmCode)`	Mencari jawaban dari parameter *question* di dalam *database*. Akan mengembalikan jawaban dari pertanyaan yang *exact-match* atau memiliki kemiripan 90% dengan parameter *question.* Jika pertanyaan yang sama atau mirip tidak ditemukan, maka akan mengembalikan rekomendasi pertanyaan yang dimaksud oleh pengguna. Rekomendasi yang diberikan adalah 3 pertanyaan dengan tingkat kemiripan diatas 45% tertinggi.

4.1.3 *File* kmp.js

Berisi fungsi-fungsi untuk melakukan pencocokan *string* dengan memanfaatkan algoritma *Knuth-Morris-Pratt*.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`function findBorder(pattern,k)`	Mengembalikan ukuran terbesar *prefix* dari *pattern*[0..k] yang juga merupakan *suffix* dari *pattern*[1..k].
2.	`function createBorderList(pattern)`	Mengembalikan *array* atau *list* berisi hasil dari fungsi pinggiran (*border function*) untuk setiap posisi pada *pattern*.
3.	`function matchKMP(text, pattern)`	Melakukan algoritma pencocokan *string* KMP untuk menemukan *pattern* pada *text*. Mengembalikan indeks *pattern* dimulai, atau -1 jika *pattern* tidak ditemukan.

4.1.4 *File* bm.js

Berisi fungsi-fungsi untuk melakukan pencocokan *string* dengan memanfaatkan algoritma *Boyer-Moore*

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`function lastOccurBuilder(pattern)`	Mengembalikan *array* atau list berisi indeks kemunculan terakhir setiap karakter pada *pattern*.
2.	`function matchBM(text, pattern)`	Melakukan algoritma pencocokan *string* BM untuk menemukan *pattern* pada *text*. Mengembalikan indeks *pattern* dimulai, atau -1 jika *pattern* tidak ditemukan.

4.1.5 *File* levenshtein-distance.js

Berisi fungsi untuk melakukan perhitungan kemiripan antara 2 buah *string*.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`function levenshteinDistance(str1, str2)`	Mengembalikan tingkat kemiripan antara 2 buah *string*.

4.1.6 *File* add.js

Berisi fungsi *wrapper* sebelum menambahkan pertanyaan ke dalam *database*.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`async function addQuery(query, algorithmCode)`	Mengambil pertanyaan dan jawaban untuk ditambahkan dari *query*, dan memanggil fungsi *addQuestion* dari processor.js

4.1.7 *File* delete.js

Berisi fungsi *wrapper* sebelum menghapus pertanyaan dari *database*.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`async function deleteQuery(query, algorithmCode)`	Mengambil pertanyaan untuk dihapus dari *query*, dan memanggil fungsi *deleteQuestion* dari processor.js

4.1.8 *File* find.js

Berisi fungsi *wrapper* sebelum mencari pertanyaan pada *database*.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`async function findQuery(query, algorithmCode)`	Mengambil pertanyaan untuk dicari dari *query*, dan memanggil fungsi *findAnswer* dari processor.js

4.1.9 *File* calculator.js

Berisi fungsi-fungsi untuk mengevaluasi suatu sintaks ekspresi matematika yang diminta oleh pengguna.

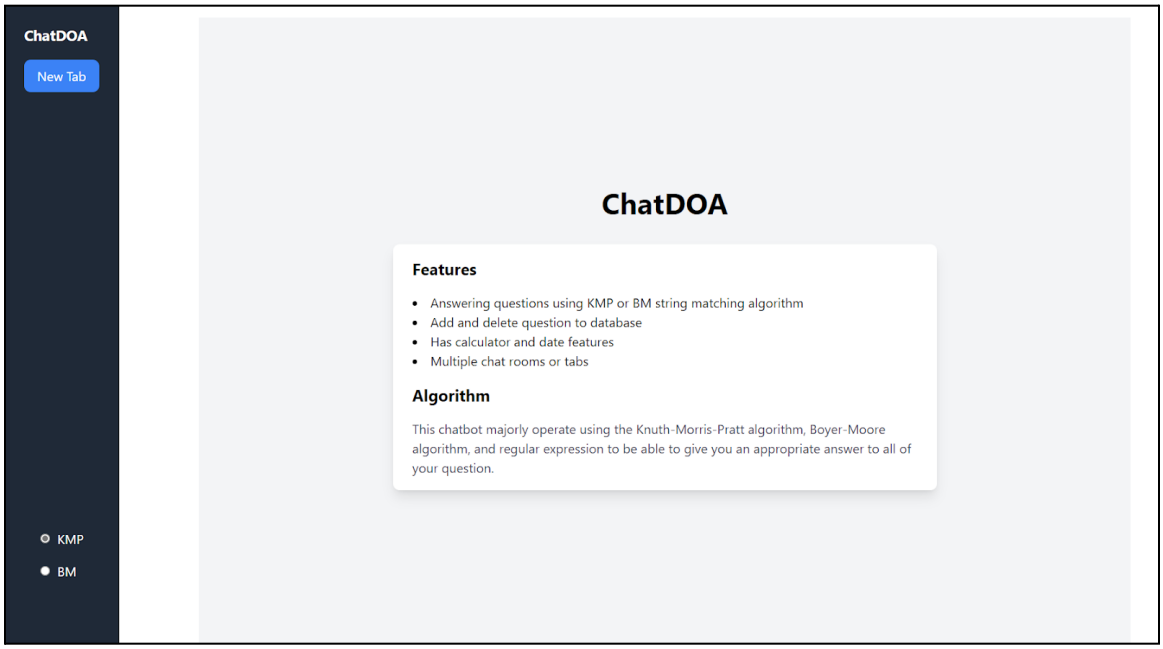
No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`function isOperand(token)`	Mengembalikan *true* jika token merupakan suatu digit. False jika tidak.
2,	`function getLenOperand(expression, start)`	Mengembalikan panjang dari operan ekspresi matematika yang dimulai dari parameter *start*
3.	`function stringToNumber(operand)`	Mengubah tipe operan dari *string* menjadi *Number*
4.	`function isOperator(token)`	Mengembalikan *true* jika token merupakan suatu operator matematika yang valid berupa +, -, *, /, dan ^.
5.	`function precedence(token)`	Mengembalikan suatu angka > 0 jika token merupakan suatu operator yang valid dengan urutan ^, * dan /, kemudian + dan -. JIka bukan operator, fungsi mengembalikan -1.
6.	`function applyOperator(operand1, operand2, operator)`	Mengembalikan nilai baru ketika *operand1* dan *operand2* dioperasikan dengan *operator*
7.	`function evaluate(expression)`	Mengembalikan hasil evaluasi ekspresi matematika yang diberikan oleh parameter *expression*. Mengembalikan *string* berupa sintaks tidak sesuai jika eskpresi matematika memiliki kesalahan.

4.1.10 *File* date.js

Berisi fungsi-fungsi untuk mencari nama hari pada tanggal yang diminta oleh pengguna.

No	*Signature* fungsi / prosedur	Tujuan fungsi
1.	`function parseDate(query)`	Mengambil angka yang merepresentasikan tanggal, bulan, dan tahun yang diminta oleh pengguna.
2,	`function getDayName(query)`	Menentukan nama hari pada tanggal yang diminta oleh pengguna jika tanggal valid. Jika tidak, mengembalikan *string* respon berupa tanggal tidak valid.

4.2 Tata Cara Penggunaan Program



Gambar 4.2.1 Halaman utama aplikasi ChatDOA

Gambar 4.2.1 memperlihatkan halaman utama dari aplikasi ChatDOA. Pada halaman utama ini, terlihat daftar fitur-fitur yang dimiliki oleh ChatDOA yaitu berupa menambahkan pertanyaan dan jawaban, menghapus pertanyaan, kalkulator, tanggal dan memberikan pertanyaan. Halaman utama juga menjelaskan beberapa algoritma *string*

matching yang dimanfaatkan oleh aplikasi. Di antaranya adalah algoritma *Knuth-Morris-Pratt, Boyer-Moore*, dan pemanfaatan *regular expression*.

Di sisi kiri halaman, terdapat sekumpulan *tab* yang merepresentasikan sebuah *room* atau *history chat* antara pengguna dengan *chatbot. Tab* baru dapat ditambahkan dengan menekan tombol *New Tab*, dan dapat juga dihapus dengan menekan ikon tempat sampah di sebelah *tab*. Di bagian ujung kiri bawah, terdapat pilihan algoritma pencocokan *string* yang dapat dipilih oleh pengguna di antara algoritma KMP atau BM.

Dalam sebuah *tab* sendiri, pengguna dapat berinteraksi dengan *chatbot* untuk memanfaatkan berbagai fitur yang dimilikinya. Terdapat 5 fitur utama yang dimiliki oleh *chatbot*. Berikut adalah daftar fitur tersebut dan bagaimana caranya agar pengguna dapat memanfaatkannya:

1. **Fitur Kalkulator**

   Pengguna dapat memanfaatkan fitur ini dengan memasukkan suatu sintaks ekspresi matematika yang hanya terdiri atas angka dan operator seperti +, -, *, /, ^, (, dan). *Chatbot* kemudian akan memberikan hasil evaluasi ekspresi matematika.

2. **Fitur Tanggal**

   Pengguna dapat memanfaatkan fitur ini dengan memasukkan format tanggal dengan format DD/MM/YYYY. *Chatbot* kemudian akan memberikan nama hari pada tanggal tersebut.

3. **Fitur Menambahkan Pertanyaan dan Jawaban**

   Pengguna dapat memanfaatkan fitur ini dengan memasukkan masukan berupa “tambahkan pertanyaan <pertanyaan> dengan jawaban <jawaban>”

4. **Fitur Pertanyaan**

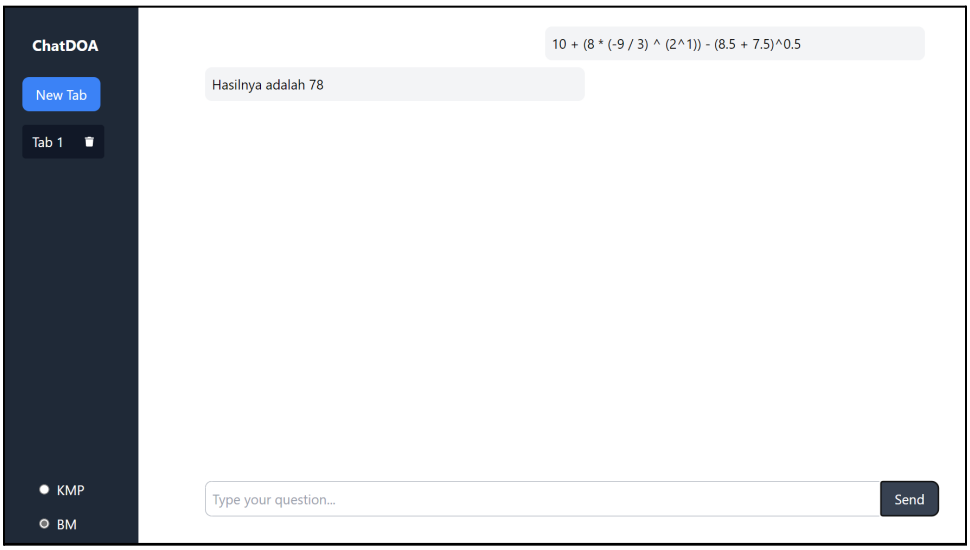
   Pengguna dapat memanfaatkan fitur ini dengan memasukkan sembarang masukan selain masukan yang digunakan untuk fitur lain.

5. **Fitur Menghapus Pertanyaan**

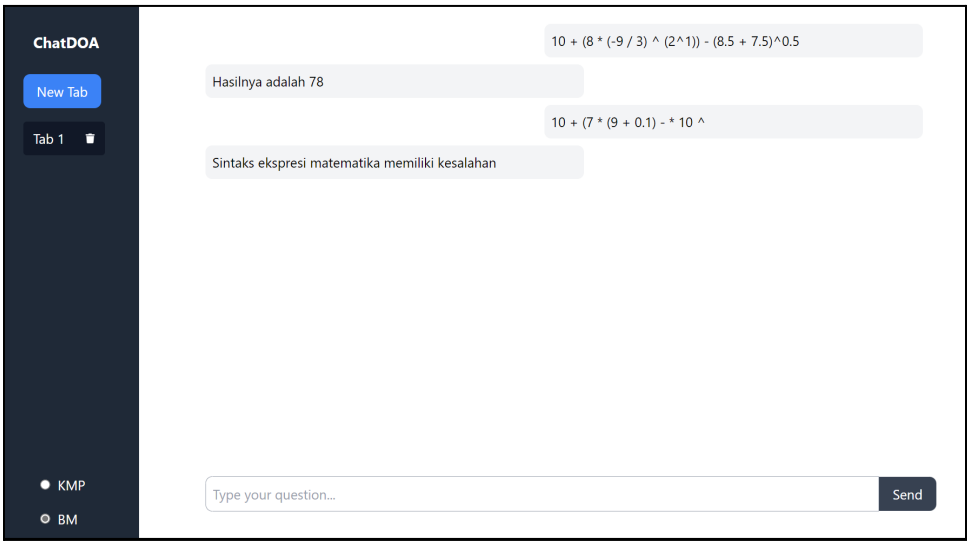
   Pengguna dapat memanfaatkan fitur ini dengan memasukkan masukan berupa “hapus pertanyaan <pertanyaan>”

4.3 Hasil Pengujian

4.3.1 Fitur Kalkulator

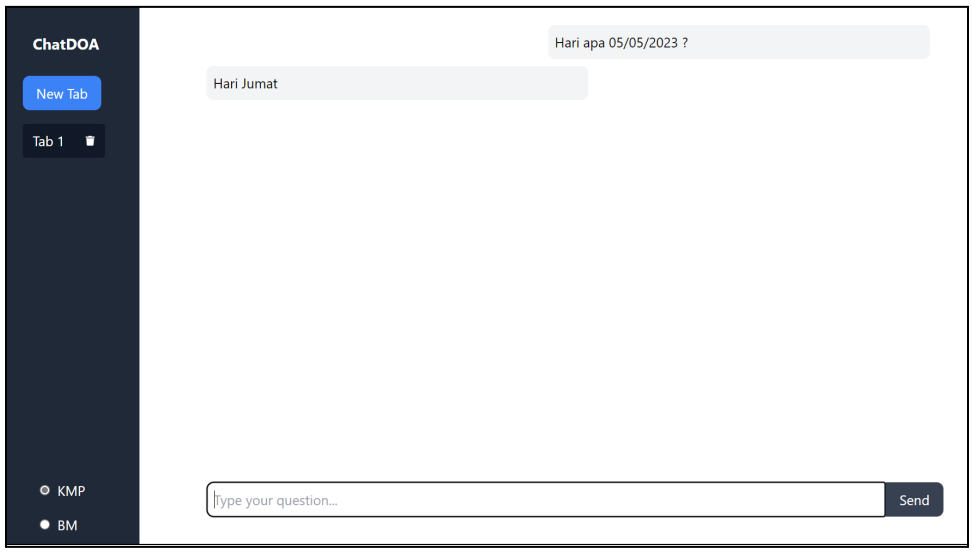


Gambar 4.3.1.1 *Chatbot* memberikan hasil evaluasi pada ekspresi matematika yang valid



Gambar 4.3.1.2 *Chatbot* memberikan pesan kesalahan pada ekspresi matematika yang tidak valid

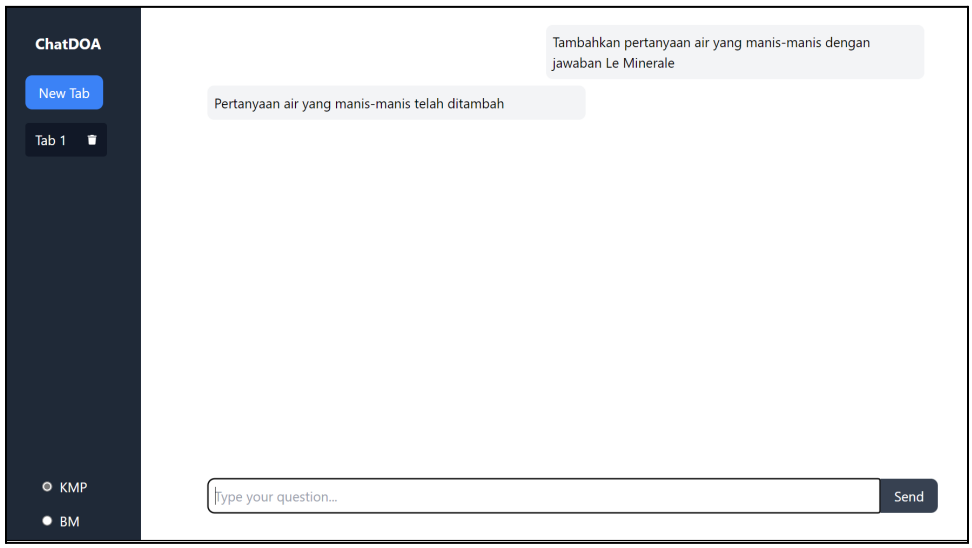
4.3.2 Fitur Tanggal



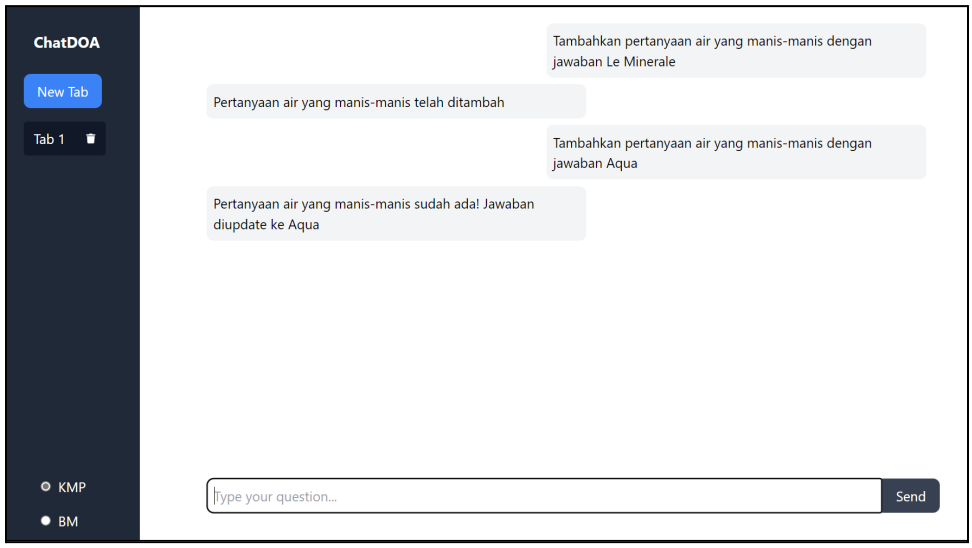
Gambar 4.3.2.1 *Chatbot* memberikan nama hari pada tanggal yang valid

Gambar 4.3.2.2 *Chatbot* memberikan pesan kesalahan pada tanggal yang tidak valid

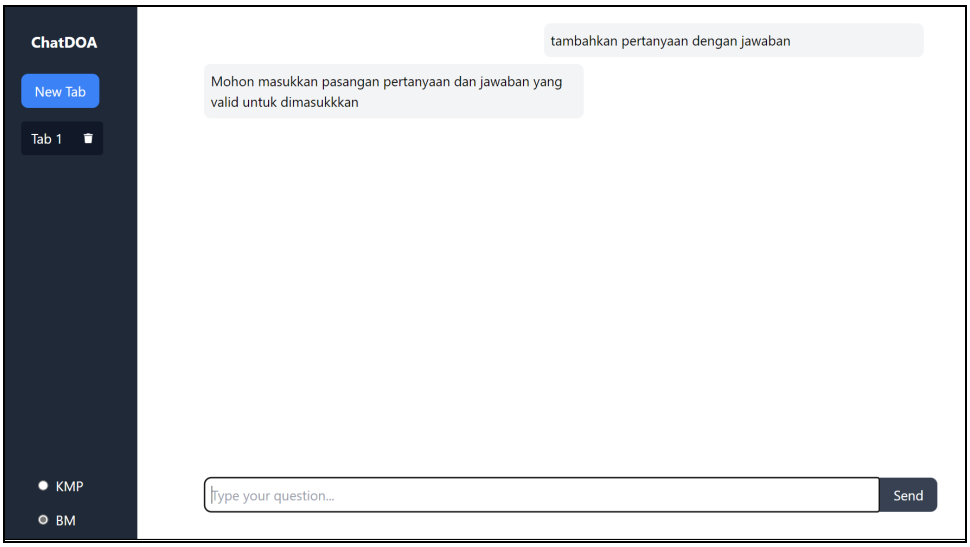
4.3.3 Fitur Menambahkan Pertanyaan dan Jawaban



Gambar 4.3.3.1 *Chatbot* memberikan pesan pertanyaan berhasil ditambah pada pertanyaan baru

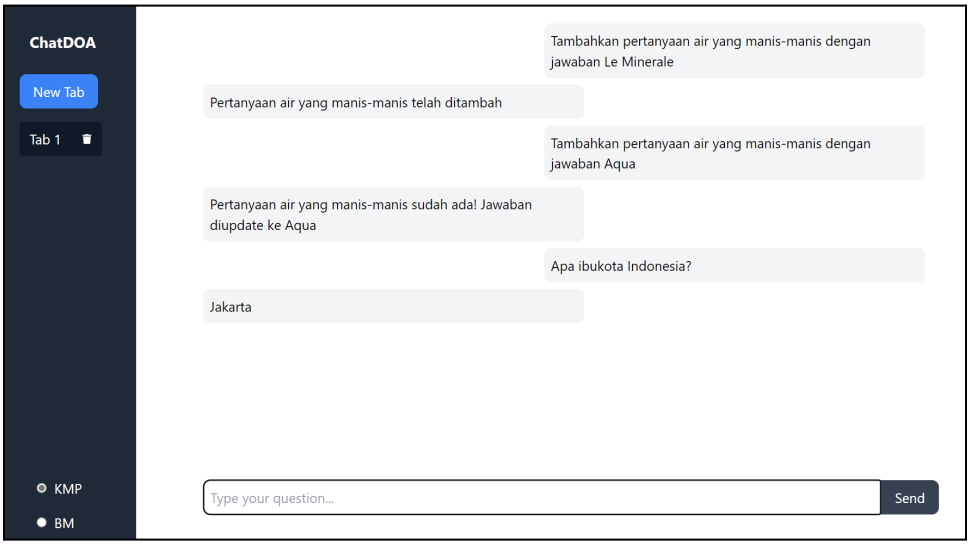


Gambar 4.3.3.2 *Chatbot* memberikan pesan pertanyaan berhasil di *update* pada pertanyaan yang sudah ada

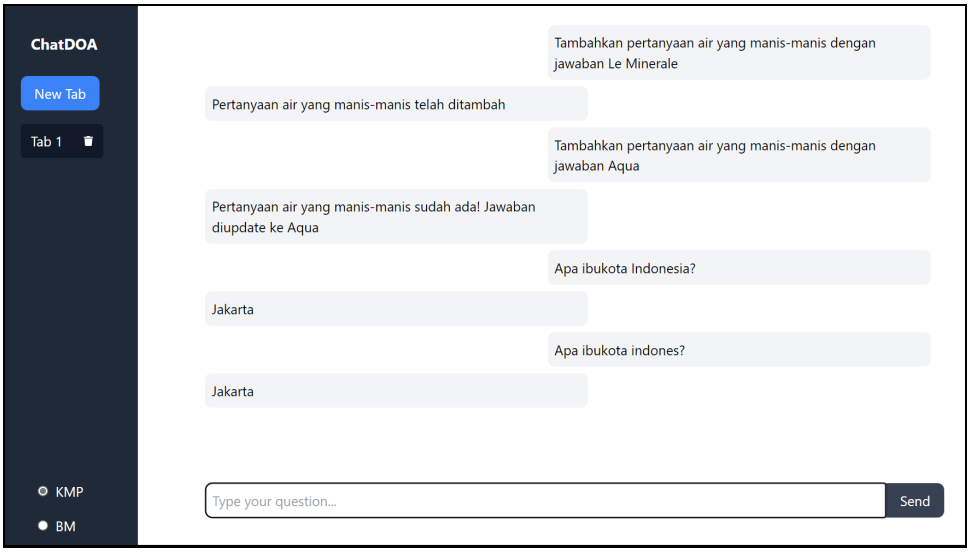


Gambar 4.3.3.3 *Chatbot* memberikan pesan kesalahan jika tidak ada pertanyaan dan jawaban untuk dimasukkan

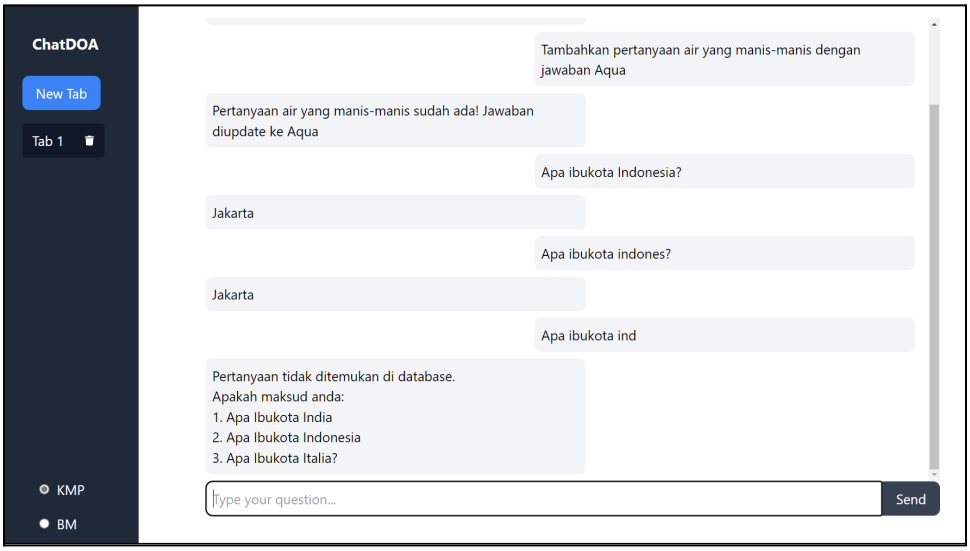
4.3.4 Fitur Pertanyaan



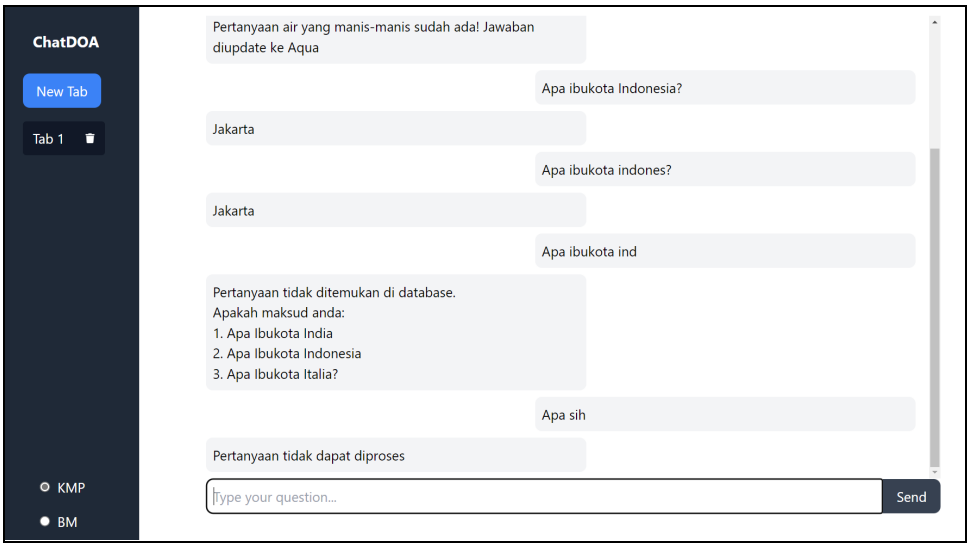
Gambar 4.3.4.1 *Chatbot* memberikan jawaban dari pertanyaan dengan kasus *exact-match*



Gambar 4.3.4.2 *Chatbot* memberikan jawaban dari pertanyaan dengan kasus kemiripan 90%

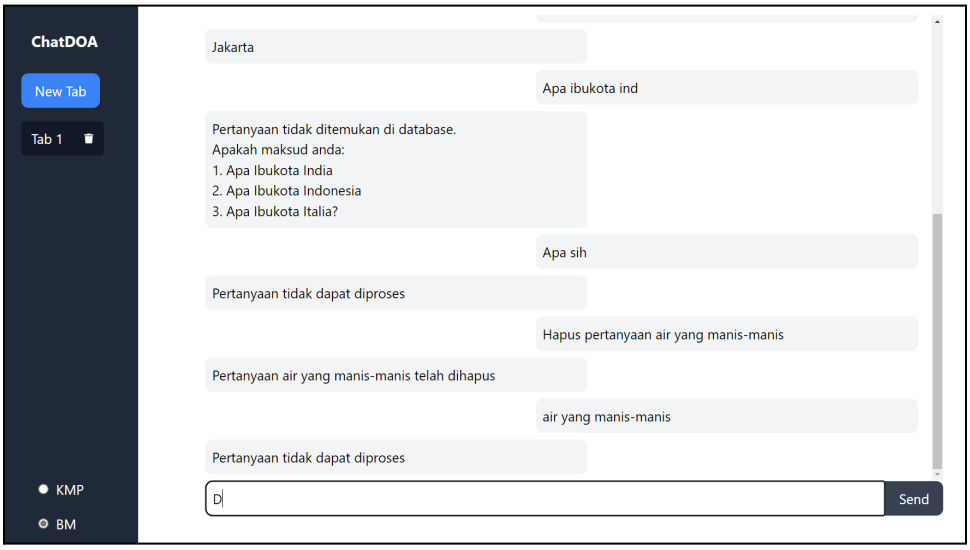


Gambar 4.3.4.3 *Chatbot* memberikan rekomendasi pertanyaan yang dimaksud dengan kasus kemiripan ≥ 45%

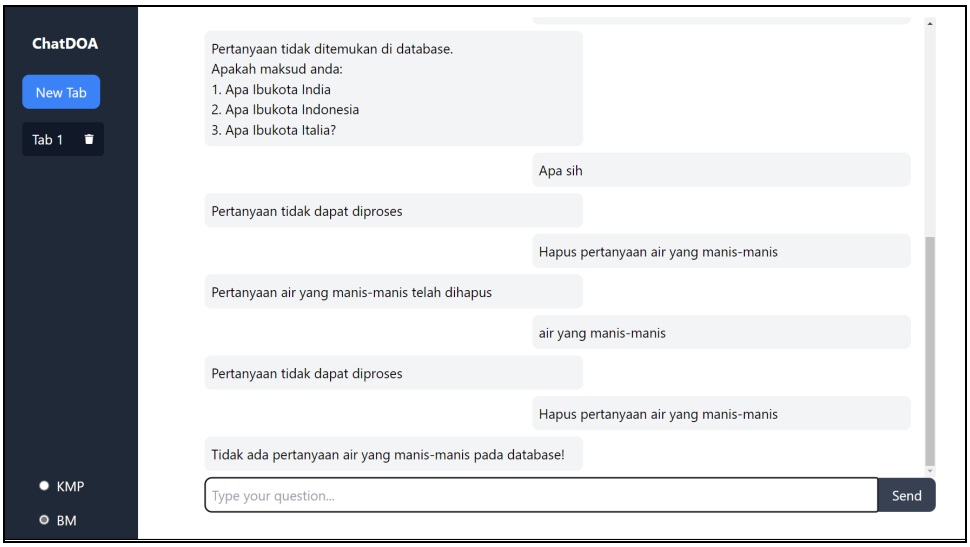


Gambar 4.3.4.4 *Chatbot* memberikan pesan pertanyaan tidak dapat diproses jika tidak ditemukan pertanyaan yang mirip

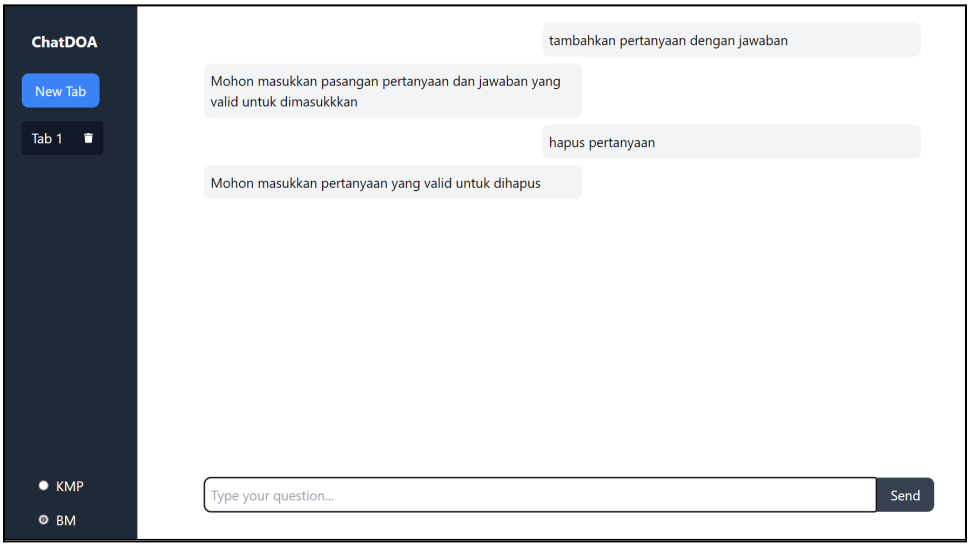
4.3.5 Fitur Menghapus Pertanyaan



Gambar 4.3.5.1 *Chatbot* memberikan pesan pertanyaan telah dihapus dari *database*, dan tidak mampu memproses pertanyaan yang telah dihapus tersebut



Gambar 4.3.5.2 *Chatbot* memberikan pesan kesalahan ketika pertanyaan yang ingin dihapus tidak ditemukan di *database*



Gambar 4.3.5.3 *Chatbot* memberikan pesan kesalahan ketika tidak ada pertanyaan yang diberikan oleh pengguna untuk dihapus

4.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian, terlihat bahwa *chatbot* telah berhasil menangani seluruh kasus-kasus yang mungkin muncul dalam setiap fitur. Kasus-kasus tersebut meliputi:

1. **Fitur Kalkulator**
   a. Sintaks ekspresi matematika valid

b. Sintaks ekspresi matematika tidak valid

2. Fitur Tanggal

a. Tanggal yang diminta valid

b. Tanggal yang diminta tidak valid

3. Fitur Menambahkan Pertanyaan dan Jawaban

a. Menambahkan pertanyaan dan jawaban baru ke *database*

b. Meng-*update* pertanyaan pada *database* dengan jawaban baru

c. Pertanyaan dan jawaban untuk ditambahkan tidak ada

4. Fitur Pertanyaan

a. Ditemukan pertanyaan yang *exact-match*

b. Ditemukan pertanyaan yang mirip 90% ke atas

c. Ditemukan pertanyaan-pertanyaan yang mirip 45% ke atas

d. Tidak ditemukan pertanyaan yang mirip

5. Fitur Menghapus Pertanyaan

a. Ditemukan pertanyaan untuk dihapus di *database*

b. Tidak ditemukan pertanyaan untuk dihapus

c. Pertanyaan untuk dihapus tidak ada

Dapat disimpulkan bahwa aplikasi telah berhasil untuk merespon setiap permintaan pengguna dengan baik dan memberikan jawaban yang tepat. Hal ini tidak terlepas dari proses perhitungan, perbandingan, dan lain-lain yang telah dijelaskan pada bagian 4.1

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma pencocokan *string* dua diantaranya yaitu *Knuth-Morris-Pratt* (KMP) dan Boyer Moore (BM). sebelum melakukan pencocokan strong dengan dua algoritma *exact match* tersebut, *string* matching dengan regular expression dilakukan. *String* yang dicocokan adalah pertanyaan dan jawaban yang dikirimkan oleh user. Pertanyaan-jawaban tersebut tersimpan dalam database, dan user dapat menambah, memodifikasi dan menghapusnya. Hasil tugas besar ini berbasis website.

5.2 Saran

Fitur-fitur yang diimplementasikan pada aplikasi ini telah berjalan dengan baik, namun terdapat ruang untuk memperbaiki aplikasi kami. Salah satunya yaitu meningkatkan bagian *user interface* sehingga dapat lebih menarik dan responsif terhadap ukuran layar. Selain itu, dapat juga ditambahkan fitur untuk mengirim pertanyaan yang lebih dari satu baris.

5.3 Komentar dan Refleksi

Setelah menyelesaikan tugas besar ini, kami menjadi lebih paham mengenai penggunaan algoritma *string matching*, khususnya algoritma KMP dan BM, dalam pembuatan aplikasi *chatbot*. Selain itu, kami juga menjadi lebih paham mengenai pengembangan aplikasi berbasis web hingga pada akhirnya kami dapat menyelesaikan dan men-*deploy* aplikasi ini dengan baik.

DAFTAR PUSTAKA

Slide Kuliah Pencocokan *String* (*String/Pattern Matching*):

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Slide Kuliah Pencocokan String dengan *Regular Expression* (*Regex*):

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf

Dokumentasi Pustaka Tailwind CSS:

https://tailwindcss.com/docs

Dokumentasi MongoDB:

https://www.mongodb.com/docs/

Lampiran

Link Repository

margarethaolivia/Tubes3_13521062 (github.com)

Link Deployment Web

ChatDOA

Link Youtube

Video Tugas Besar 3 IF2211 - Strategi Algoritma 2022/2023 Kelompok ChatDOA - YouTube