LAPORAN TUGAS BESAR III

Penerapan String Matching dan Regular Expression dalam Pembuatan ChatGPT Sederhana

Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan Mata Kuliah Strategi Algoritma (IF2211)

Disusun oleh:
Kelompok 36
“geepeetee”

Anggota:
Muhammad Farrel Danendra Rachim (13521048)
Fazel Ginanda (13521098)
Zidane Firzatullah (13521163)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER II TAHUN 2022/2023

Daftar Isi

BAB 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)**. **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.

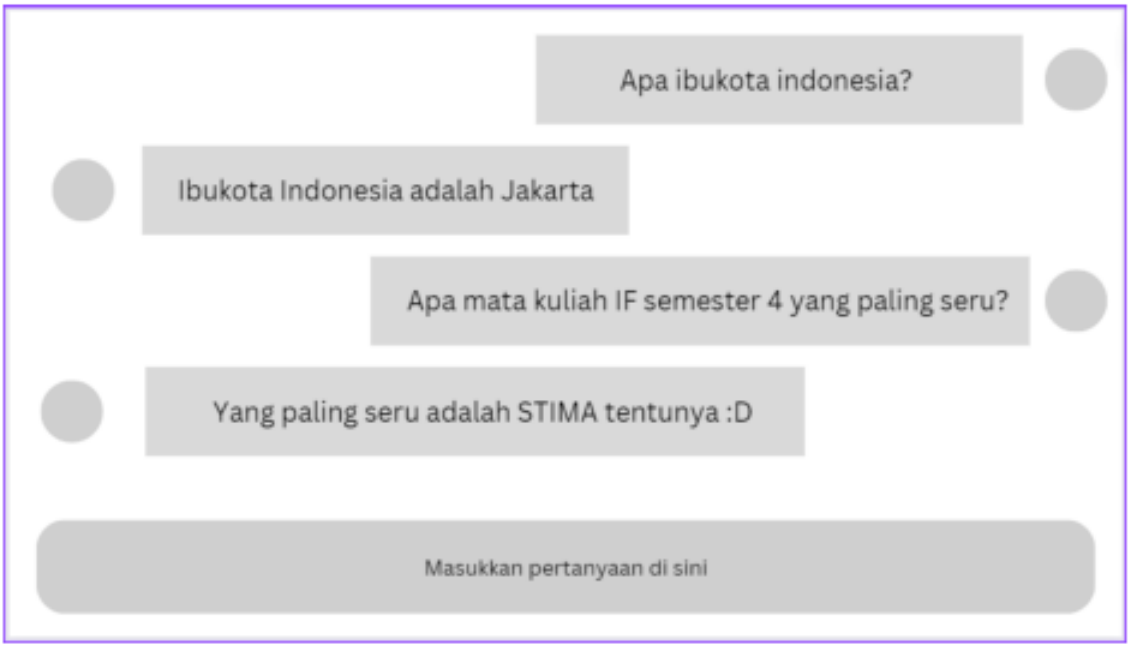
4. **Tambah pertanyaan dan jawaban ke database**

   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 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**)**

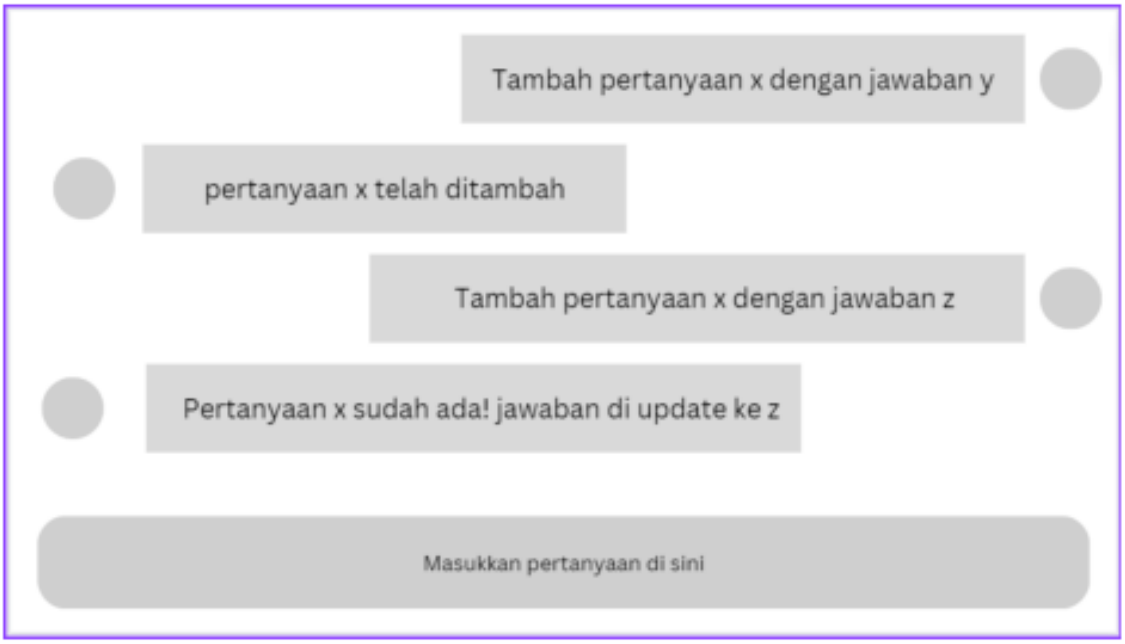


Gambar 2. Ilustrasi Fitur Pertanyaan teks kasus exact

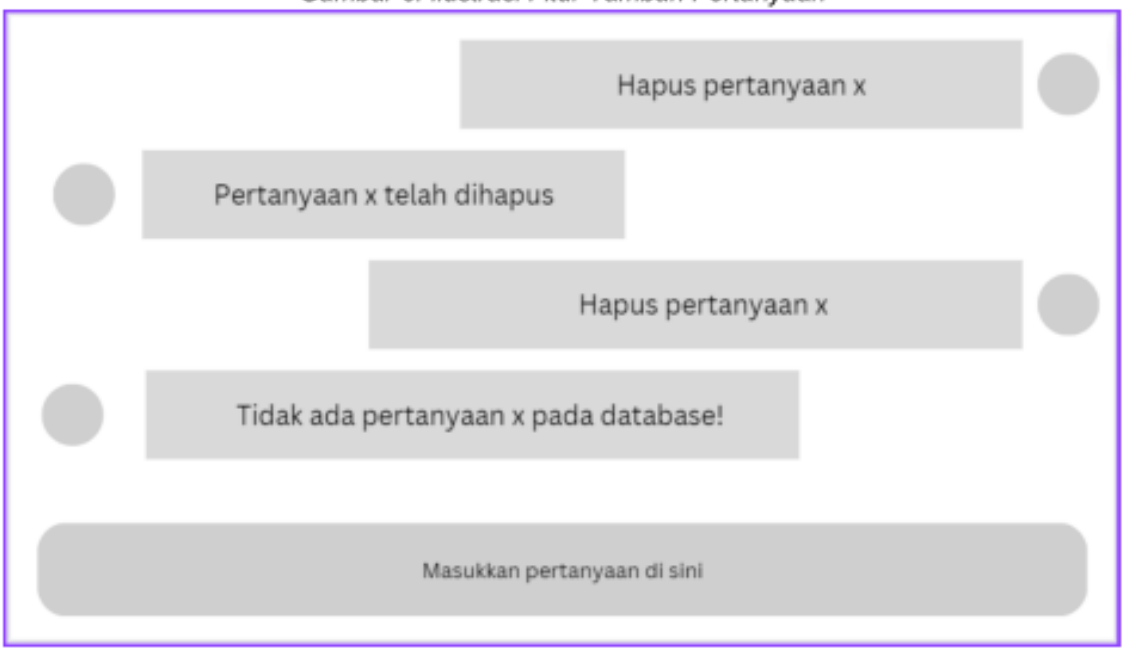
Gambar 3. Ilustrasi Fitur Pertanyaan teks kasus tidak exact

Gambar 4. Ilustrasi Fitur Kalkulator

Gambar 5. Ilustrasi Fitur Tanggal



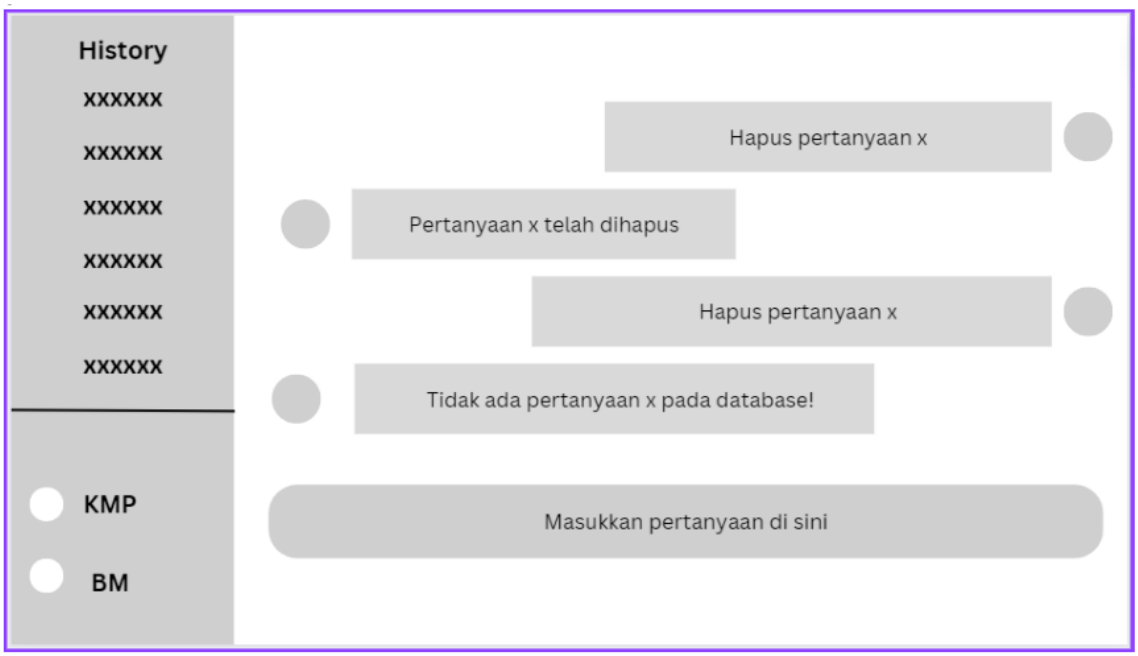
Gambar 6. Ilustrasi Fitur Tambah Pertanyaan



Gambar 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 merestore seluruh pertanyaan dan jawaban di halaman utama. Contoh ilustrasi keseluruhan:



Gambar 8. Ilustrasi Keseluruhan

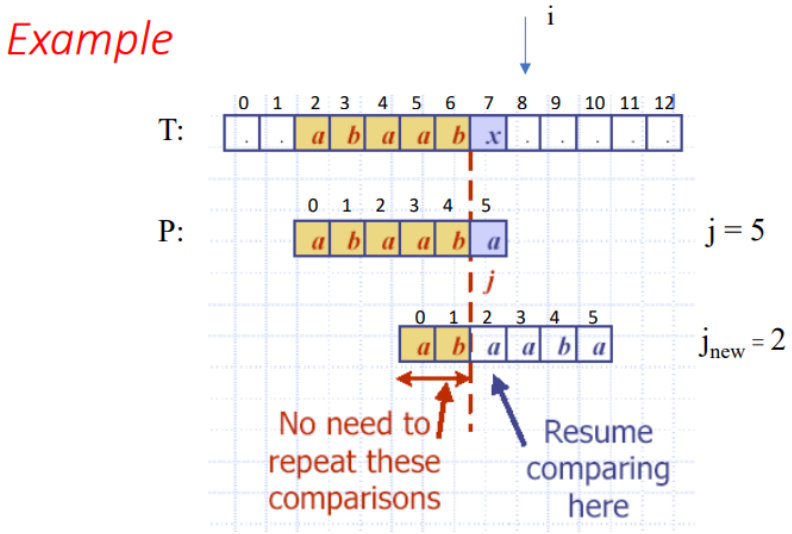
BAB 2
Landasan Teori

2.1. Dasar Teori

2.1.1. Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah sebuah algoritma *string matching* yang mencari pola di dalam teks dengan urutan dari kiri ke kanan. Hal ini dilakukan dengan menghindari perbandingan dengan elemen "S" yang sudah terlibat dalam perbandingan di pola "p" untuk dicocokkan. Dengan kata lain, algoritma ini tidak pernah melakukan *backtracking* terhadap string "S". Jika algoritma menemukan *mismatch*, algoritma dapat bergeser sebanyak satu *space* atau lebih.

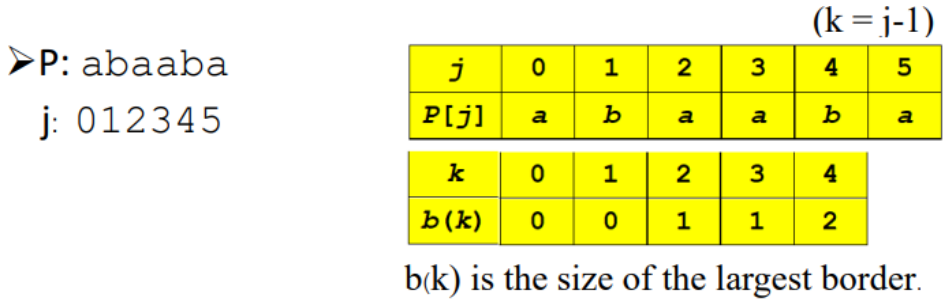
Jika terdapat *mismatch* antara teks T dan pola P pada P[j] (T[i] ≠ P[j]), agar menghindari perbandingan yang tidak diperlukan, pergeseran terbesar yang dapat dilakukan yaitu *prefix* terbesar dari P[0..j-1] yang merupakan *suffix* dari P[1..j-1]. Berikut salah satu contoh ilustrasi algoritma KMP:



Gambar 2.1. Contoh Algoritma KMP (Sumber: PPT Pak Rinaldi Munir)

KMP memiliki *Border Function* b(k) (atau *Failure Function*) yang melakukan *preprocessing* terhadap pola untuk mencari *prefix* dari pola P yang cocok dengan pola P itu sendiri. *Border function* didefinisikan sebagai ukuran prefix terbesar dari P[0..k] yang juga merupakan suffix dari P[1..k]. *Border function* direpresentasikan sebagai sebuah tabel dengan j adalah indeks pola P, P[j] huruf abjad P pada posisi j, k yang bernilai j - 1, dan b[k] sebagai hasil *Border Function* pada k. Dalam kode, b() direpresentasikan

sebagai sebuah *array* yang dibuat mirip seperti tabel di bawah. Dalam tabel di bawah, b(4) bernilai 2 karena diketahui ukuran "ab" adalah 2, dan sedang dicari ukuran prefix terbesar dari "abaab" (k = 4, [0..k]) yang merupakan suffix dari "baab" (k = 4, [1..k]).



Gambar 2.2. Contoh *Border Function* (Sumber: PPT Pak Rinaldi Munir)

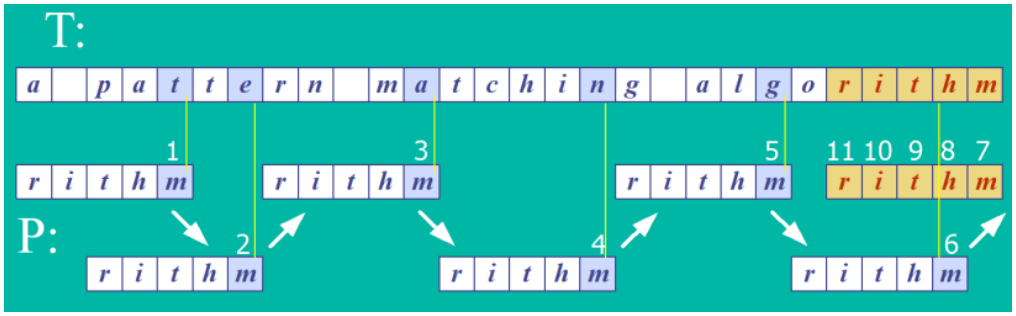
Dengan bantuan *border function*, algoritma KMP memodifikasikan algoritma *brute-force*. Jika ditemukan *mismatch* pada P[j] (T[i] ≠ P[j]), lakukan k = j - 1, dan peroleh j baru dengan j = b(k).

Kompleksitas algoritma KMP adalah O(m + n), dengan O(m) dihasilkan dari perhitungan *border function*, dan O(n) dihasilkan dari pencarian *string*. Algoritma KMP sangat menguntungkan dibandingkan dengan algoritma lain seperti *brute force* karena KMP tidak perlu bergerak mundur saat melakukan pencocokkan pola di teks T, yang sangat berguna khususnya jika file berukuran besar. Namun, kelemahannya yaitu kecenderungan *mismatch* akan meningkat saat ukuran alfabet juga meningkat, sehingga kompleksitasnya lebih mendekati *brute force*.

2.1.2. Boyer-Moore (BM)

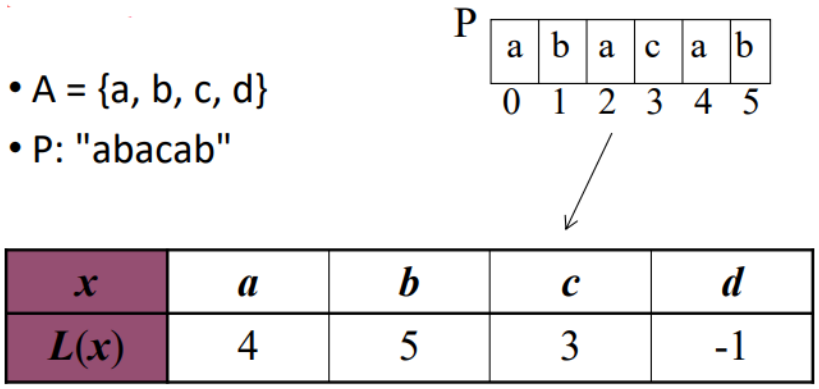
Algoritma Boyer-Moore adalah sebuah algoritma *string matching* yang didasarkan pada dua teknik: a) *Looking Glass Technique*, yaitu pencarian P di dalam T dari kanan ke kiri (*backwards*) melalui P, dimulai dari ujung P; b) *Character-Jump Technique*, yaitu penentuan seberapa banyak P akan bergeser tergantung posisi terakhir karakter berdasarkan posisi di P. Hal ini terjadi saat ada *mismatch* di T[i] == x, atau karakter di pola P[j] yang tidak sama dengan T[i]. Terdapat tiga buah kasus untuk melakukan teknik ini:

- Jika P mengandung x, geser P ke kanan untuk menyelaraskan kemunculan x terakhir di P dengan T[i].
- Jika P mengandung x, namun pergeseran ke kanan ke kemunculan terakhir x tidak mungkin, geser P ke kanan sebanyak satu karakter ke T[i+1].
- Jika kedua kasus di atas tidak berlaku, geser P untuk menyelaraskan P[0] dengan T[i+1].



Gambar 2.3. Contoh Implementasi Algoritma BM (Sumber: PPT Pak Rinaldi Munir)

Algoritma BM memiliki *Last Occurrence Function* (L(x)), yang melakukan *preprocessing* pola P dan abjad A dengan memetakan semua huruf di A menjadi integer. Dengan x sebuah huruf di A, L(x) didefinisikan sebagai indeks terbesar i sehingga P[i] == x, sedangkan jika tidak ada indeks yang memenuhi syarat tersebut, nilainya -1. Dalam kode BM, L() dihitung saat pola P merupakan *read in*.



x	*a*	*b*	*c*	*d*
L(*x*)	4	5	3	-1

Gambar 2.4. Contoh Implementasi *Last Occurrence Function* (Sumber: PPT Pak Rinaldi Munir)

2.1.3. Regular Expression

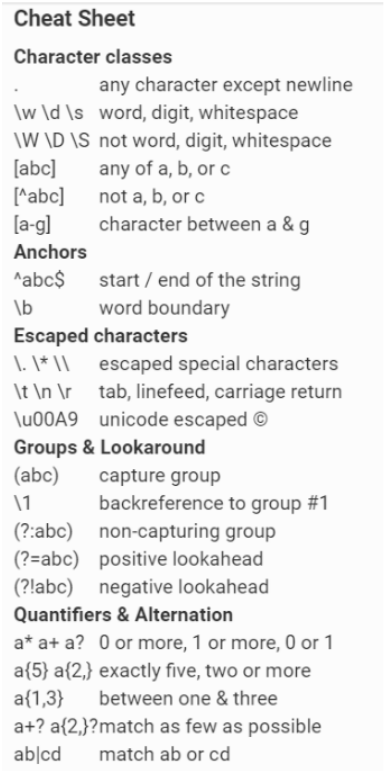
Regular Expression atau regex adalah notasi standar yang mendeskripsikan suatu pola (pattern) berupa urutan karakter atau string. Regex digunakan untuk pencocokan string (string matching) dengan efisien.

Pencarian string literal adalah bentuk paling dasar dari regex. Misalkan pencarian regex “hai” di teks “hai halo hai” akan menghasilkan dua buah pola “hai” di teks tersebut.

Regex memiliki metakarakter yang diinterpretasikan secara khusus oleh mesin regex. Beberapa contohnya yaitu:

- Kurung siku yang menentukan kumpulan karakter yang ingin dicocokkan. Misal kurung [abcde] atau [a-e] akan cocok jika string memiliki a, b, c, d, atau e.
- Tanda titik mencocokkan sebuah karakter. Misal regex ".udi" pada string "udi Budi Rudi" akan menghasilkan pencocokkan "Budi Rudi".
- Simbol ^ digunakan untuk mengecek jika sebuah string dimulai dengan karakter tertentu. Misal "^a" pada string "abc" akan menghasilkan sebuah pencocokkan.
- Simbol $ digunakan untuk mengecek jika sebuah string diakhiri dengan karakter tertentu. Misal "^a" pada string "formula" akan menghasilkan sebuah pencocokkan.
- Simbol * mencocokkan nol atau lebih kemunculan pola di sebelah kirinya.
- Simbol + mencocokkan satu atau lebih kemunculan pola di sebelah kirinya.
- Simbol ? mencocokkan nol atau satu kemunculan pola di sebelah kirinya
- Simbol { }: Dalam regex a{n, m}, di mana m dan n merupakan sebuah angka dan n <= m, terdapat pengulangan pola di sebelah kiri *curly bracket* (dalam contoh ini "a") di antara nilai n dan m.
- Simbol |: Regex a | b mencocokkan string apapun yang mengandung a atau b.
- Simbol () digunakan untuk mengelompokkan subpola. Contohnya, (a|b|c)xz mencocokkan string apapun yang memiliki a, b, atau c yang diikuti oleh xz.
- Simbol backslash \: Meng-*escape* karakter apapun termasuk metakarakter.
- Terdapat juga *special sequences* seperti \b, \d, \s, \w, dan seterusnya untuk memudahkan penulisan pola.

Berikut *cheat sheet* yang dapat digunakan untuk memudahkan penggunaan regex:



Cheat Sheet

Character classes

.	any character except newline
\w \d \s	word, digit, whitespace
\W \D \S	not word, digit, whitespace
[abc]	any of a, b, or c
[^abc]	not a, b, or c
[a-g]	character between a & g

Anchors

^abc$	start / end of the string
\b	word boundary

Escaped characters

\. * \\	escaped special characters
\t \n \r	tab, linefeed, carriage return
\u00A9	unicode escaped ©

Groups & Lookaround

(abc)	capture group
\1	backreference to group #1
(?:abc)	non-capturing group
(?=abc)	positive lookahead
(?!abc)	negative lookahead

Quantifiers & Alternation

a* a+ a?	0 or more, 1 or more, 0 or 1
a{5} a{2,}	exactly five, two or more
a{1,3}	between one & three
a+? a{2,}?	match as few as possible
ab|cd	match ab or cd

Gambar 2.5. *Cheat Sheet* Penggunaan Regex (Sumber: https://www.regexpal.com/)

2.2. Aplikasi Web

Sebagai implementasi tugas besar ini, kami membuat aplikasi berbasis web dengan bahasa Golang yang membentuk *backend* dan bahasa React.js sebagai penyusun *frontend*, serta penyimpanan data menggunakan MySQL. Dengan aplikasi ini, pengguna dapat:

- Berinteraksi dengan chatbot dengan mengirim sebuah pertanyaan. Chatbot akan menyesuaikan pertanyaan yang dikirim dengan query yang sudah disimpan di dalam database.
- Melakukan operasi matematika sederhana seperti tambah, kali, pangkat.
- Mencari tahu hari apa di suatu tanggal dengan mengirim tanggal dengan format hari/bulan/tahun.
- Menambah pertanyaan serta jawaban dari pertanyaan tersebut ke database dengan format query: “Tambahkan pertanyan xxx dengan jawaban yyy”.
- Menghapus pertanyaan dari database dengan format query: “Hapus pertanyaan xxx”.

BAB 3
Analisis Pemecahan Masalah

3.1. Langkah-langkah Pemecahan Masalah

Berikut adalah langkah-langkah pemecahan masalah interaksi dengan chatbot dengan algoritma KMP.

1. Membuat fungsi SearchKMP yang menerima pattern yang akan dicari dan teks yang akan dicocokkan dengan pattern tersebut.
2. Membuat empat variabel untuk menyatakan state pada proses pencocokan, yaitu j, idx, inMatch, dan found. Variabel j menyatakan indeks terakhir dari pattern yang cocok dengan teks. Variabel idx menyatakan indeks dari teks. Variable inMatch bertipe boolean yang bernilai true jika ditemukan kecocokan karakter antara pattern dengan teks. Variabel found bertipe boolean yang bernilai true jika telah ditemukan kecocokan antara seluruh karakter pada pattern dengan sekumpulan karakter pada teks.
3. Jika jumlah karakter pada pattern melebihi jumlah karakter pada teks, maka fungsi Search kmp mengembalikan nilai -1.
4. Jika tidak demikian, dilakukan iterasi terhadap teks.
5. Pada setiap iterasi, dilakukan pengecekan terhadap karakter pada pattern dan karakter pada teks. Jika ditemukan kecocokan antara karakter pada pattern dengan karakter pada teks, maka variabel inMatch di-*assign* dengan nilai true. Pada iterasi selanjutnya, selama ditemukan kecocokan antara karakter pattern dengan karakter teks, maka nilai j akan selalu di-increment. Hal ini bermakna bahwa indeks terakhir dari pattern yang cocok dengan teks berubah atau bergeser.
6. Jika pada salah satu tahap iterasi, ditemukan ketidakcocokan antara karakter pada pattern dengan karakter pada teks, maka akan dipanggil fungsi getJumpArr yang bertugas sebagai *border function.* Fungsi ini mengembalikan nilai j yaitu indeks pertama pada pattern untuk dilakukan pencocokan setelah ditemukan ketidakcocokan pada teks dan setelah pergeseran pattern.
7. Jika seluruh karakter pada pattern cocok dengan dengan sekumpulan karakter pada teks, maka fungsi SearchKMP mengembalikan indeks pada teks yang menjadi indeks pertama dari kemunculan pattern tersebut.
8. Jika pada seluruh tahap iterasi tidak ditemukan pattern pada teks, maka fungsi SearchKMP mengembalikan nilai -1.

Berikut adalah langkah-langkah pemecahan masalah interaksi dengan chatbot dengan algoritma BM.

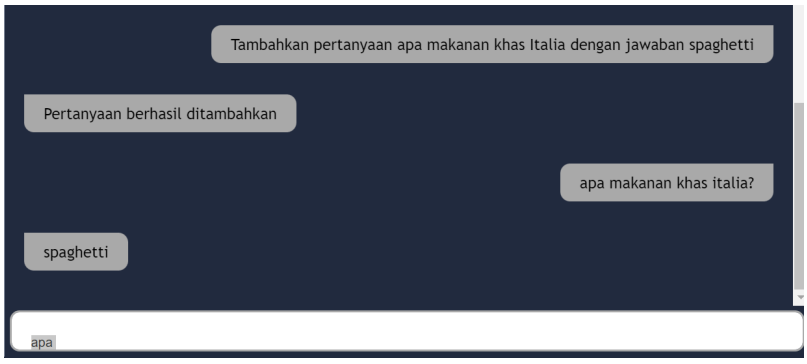
1. Membuat fungsi SearchBM yang menerima pattern dan teks.
2. Membuat tabel charShiftTable dengan memanggil fungsi getCharShiftTable yang memetakan nilai ASCII dari karakter menjadi indeks kemunculan terakhir dari sebuah karakter pada pattern.
3. Membuat empat variabel lokal, yaitu textLen, patternLen, j, dan idx. Variabel textLen menyatakan jumlah karakter dari teks. Variabel patternLen menyatakan jumlah karakter dari pattern. Variabel j menyatakan indeks dari pattern. Variabel idx menyatakan indeks dari teks.
4. Jika jumlah karakter pada pattern lebih banyak daripada jumlah karakter pada teks, maka fungsi mengembalikan nilai -1;
5. Jika tidak demikian, maka dilakukan proses pencocokan melalui iterasi terhadap teks.
6. Pada setiap tahap iterasi, dilakukan pencocokan karakter pada pattern dengan karakter pada teks. Pencocokan dimulai pada karakter terakhir dari pattern.
7. Jika pada salah satu tahap iterasi ditemukan kecocokan antara karakter pattern dengan karakter teks, maka dilakukan *looking glass technique,* yaitu mengurangi idx dan j dengan satu. Dengan kata lain, pattern tidak digeser dan proses pencocokan atau pengecekan kesamaan karakter mundur sebanyak satu karakter. Jika kesamaan ini terus ditemukan pada tahap-tahap iterasi selanjutnya sehingga seluruh karakter pada pattern cocok dengan sekumpulan karakter pada teks dan nilai j sudah mencapai nilai -1, maka iterasi berakhir dan fungsi mengembalikan nilai dari idx ditambah satu. Alasan pengurangan ini adalah karena pada tahap iterasi terakhir, nilai idx sudah dikurangi satu. Oleh karena itu, untuk mendapatkan indeks karakter pertama dari kemunculan pattern pada teks, nilai idx harus ditambah satu agar didapatkan nilai sebenarnya.
8. Jika pada salah satu tahap iterasi, karakter pada pattern tidak sama dengan karakter pada teks, maka dilakukan *character-jump technique.* Pada teknik ini, dilakukan pembacaan nilai pada tabel charShiftTable yang berkoresponden dengan karakter teks yang tidak cocok dengan pattern. Hasil pembacaan ini disimpan dalam variabel ni. Kemudian, nilai idx diperbarui dengan nilai idx yang lama ditambah dengan panjang pattern ditambah dengan nilai minimum dari j dan ni. Setelah itu, nilai j dikembalikan lagi menjadi nilai semula, yaitu indeks terakhir dari pattern. Hal ini berarti pada pencocokan selanjutnya setelah pergeseran, karakter pertama dari pattern yang dicocokkan dengan karakter teks adalah karakter terakhir.

9. Jika iterasi pada teks berakhir dan nilai j tidak sama dengan -1 yang mengimplikasikan pattern tidak ditemukan pada teks, maka fungsi SearchBM mengembalikan nilai -1.

3.2. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun

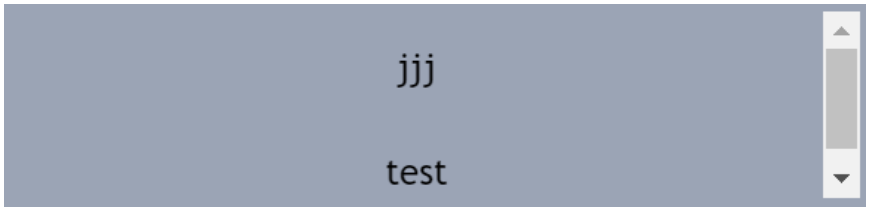
Berikut adalah fitur fungsional yang dimiliki oleh aplikasi web kami:

- Main page
    - Kolom chat
      Menyediakan fitur chat kepada pengguna kepada bot. Di sini pengguna bisa menanyakan bot pertanyaan, menambah pertanyaan dan menghapus pertanyaan dari database. Untuk mengirim pertanyaan, pengguna cukup menekan enter.



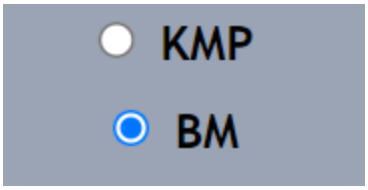
Gambar 3.2.1 Kolom chat

    - Tab history
      Tab ini menyimpan seluruh percakapan pengguna dengan bot yang telah diakses pada sebuah sesi tertentu. Satu sesi berlangsung sampai sebuah tab baru aplikasi dibuat, tab ditutup, atau tab di-refresh.



Gambar 3.2.2 Tab history

    - Radio algorithm button
      Pengguna dapat memilih algoritma mana yang digunakan: KMP atau BM.



Gambar 3.2.3 Radio algorithm button

BAB 4
Implementasi dan Pengujian

4.1. Spesifikasi Teknis Program

Untuk mengirim pesan dari pengguna dan pemerolehan data dari basis data, program memakai file .json, dengan file dan contoh format sebagai berikut:

```
question = {
   "question": "ping",
   "search_algorithm": "kmp",
   "session_id": "7698f4895b"
}
history = {
        “sessionName”={item.SessionName},
        “sessionId”={item.SessionID},
        “data”={data},
        “setData”={setData}
}
history/sessionId = {
   history_id = “sjsjajskasjk”,
   session_id = “7698f4895b”,
   user_entry = “ping”,
   answer = “pong”,
   creation_date = “creation_date”
}
```

Berikut adalah spesifikasi teknis program kami yang berada di folder backend/internal/algorithm.

a. kmp.go (implementasi algoritma KMP)

Struktur Data dan Fungsi	Penjelasan Singkat
SearchKMP()	Mencari indeks pertama pada teks yang cocok dengan pattern dengan algoritma Knuth-Morris-Pratt
getJumpArr()	Mencari indeks pada pattern yang menjadi indeks pertama pencocokan selanjutnya setelah pergeseran pattern atau sebagai fungsi pinggiran

b. boyer-moore.go (implementasi algoritma BM)

Struktur Data dan Fungsi	Penjelasan Singkat
SearchBMP()	Mencari indeks pertama pada teks yang cocok dengan pattern dengan algoritma Boyer-Moore
getCharShiftTable()	Memetakan nilai ASCII dari karakter pada pattern menjadi indeks terakhir dari kemunculan karakter tersebut pada pattern.

c. Calculator.go (implementasi algoritma kalkulator)

Struktur Data dan Fungsi	Penjelasan Singkat
CalculateExpression()	Mengevaluasi ekspresi aritmatika dan mengembalikan hasil dari evaluasi tersebut atau pesan kesalahan
getPrec()	Menentukan urutan presedensi dari operator aritmatika
basicCalc()	Menghitung nilai dari ekspresi paling sederhana yaitu dua operan dan satu operator

d. dateAI.go (implementasi algoritma tanggal)

Struktur Data dan Fungsi	Penjelasan Singkat
GetDayDate()	Menentukan hari dari tanggal dalam format string

getX()	Menerima tanggal dalam format string menjadi tanggal relatif terhadap tanggal pertama dalam satu tahun kalender

e. lev_distance.go (implementasi Levenshtein distance)

Struktur Data dan Fungsi	Penjelasan Singkat
GetLevDistance()	Menghitung tingkat kemiripan pertanyaan yang diberikan pengguna dengan pertanyaan yang ada pada database

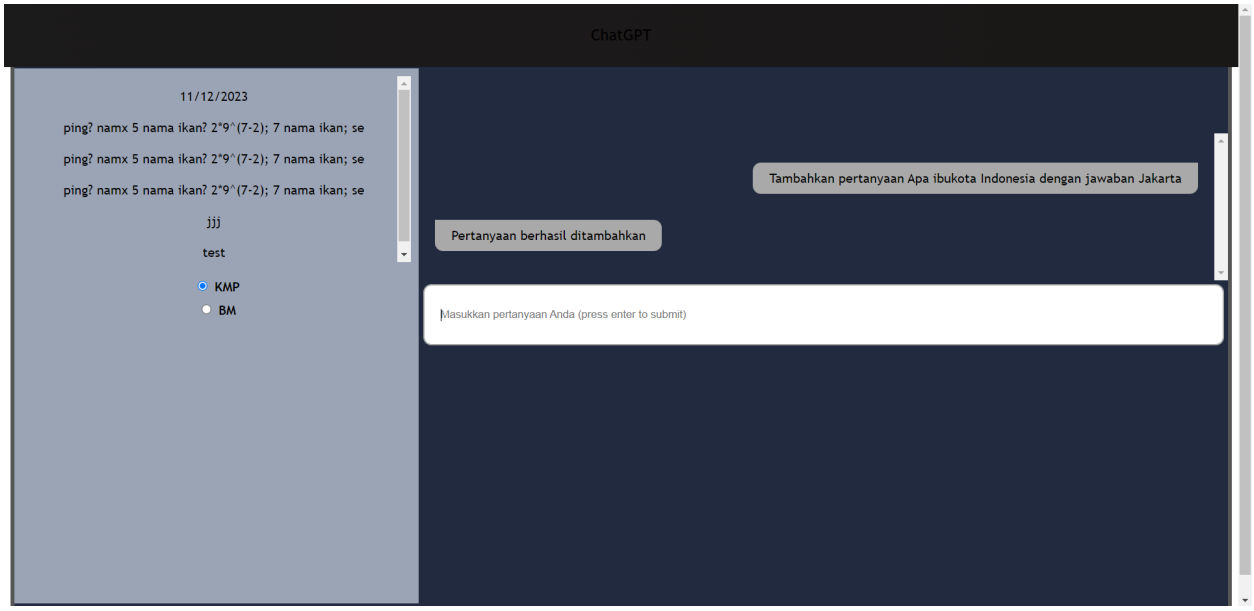
4.2. Tata Cara Penggunaan Program

Program dibangun dengan bahasa Reactjs untuk frontend dan Golang untuk backend. Kami telah mendeploy program kami ke web, link ada di lampiran di akhir dokumen. Namun, jika ingin menjalankan program dari kode *repository* (yang juga sudah tersedia di lampiran).

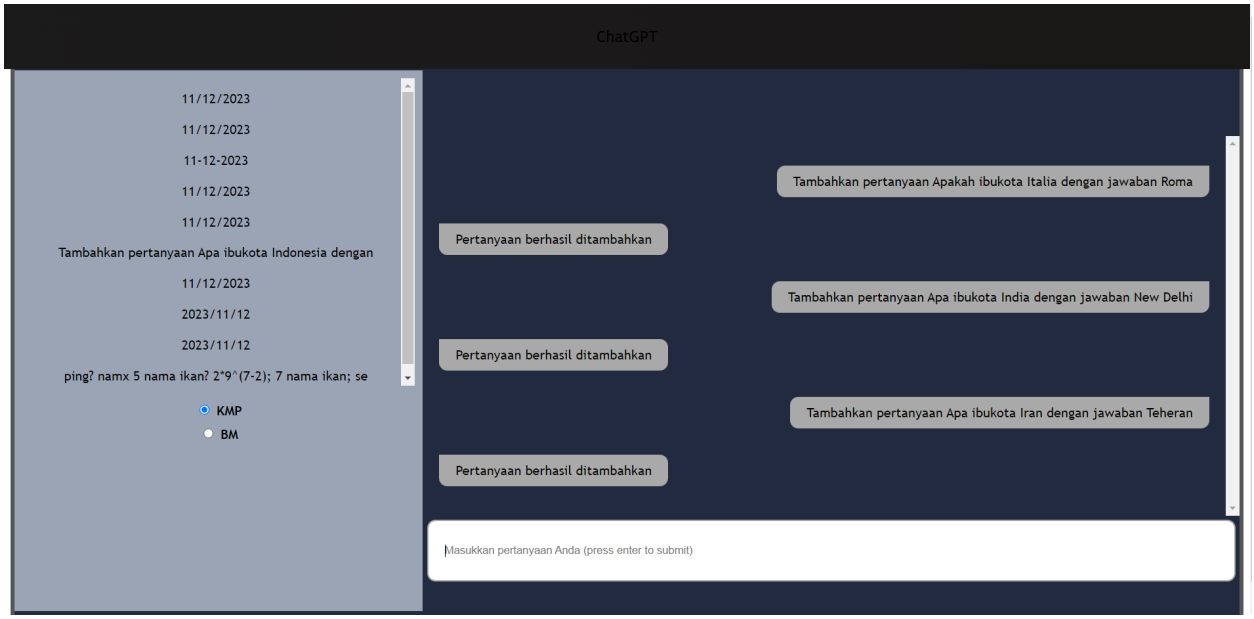
1. Install golang di komputer Anda.
2. Buka terminal dalam folder, ketik “cd src/backend”
3. Ketik “go mod vendor” yang membangun direktori bernama “vendor” di root directory modul utama.
4. Ketik “cd app”
5. Ketik “go build” untuk menjalankan program
6. Ketik “./app”

4.3. Hasil Pengujian

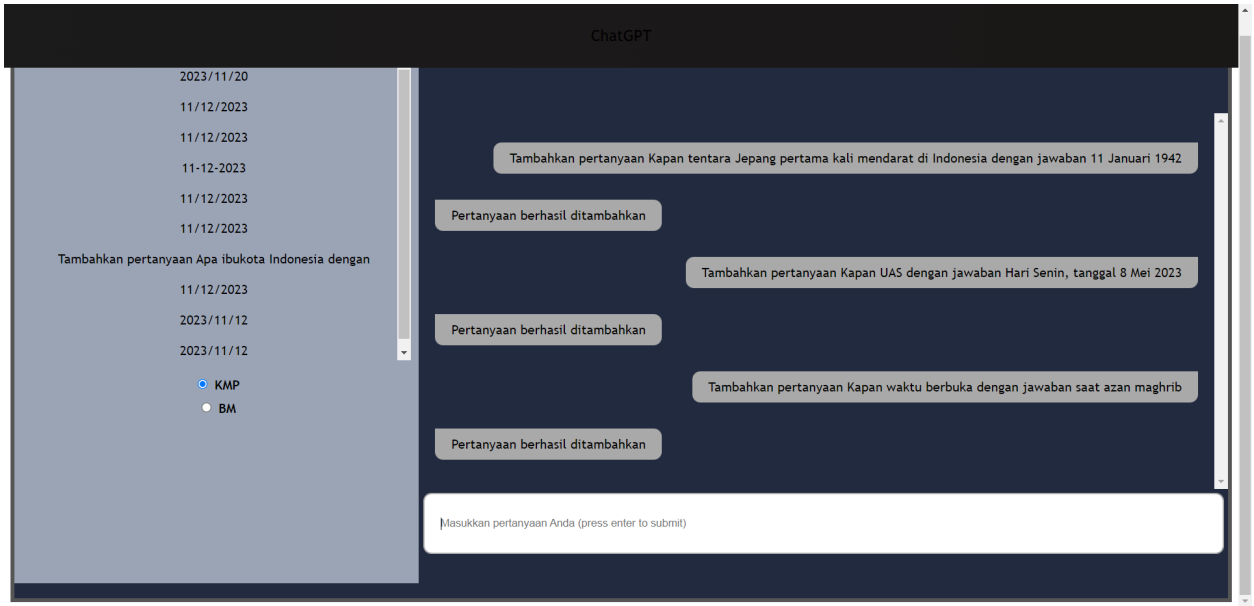
4.3.1 Menambahkan Pertanyaan ke Database



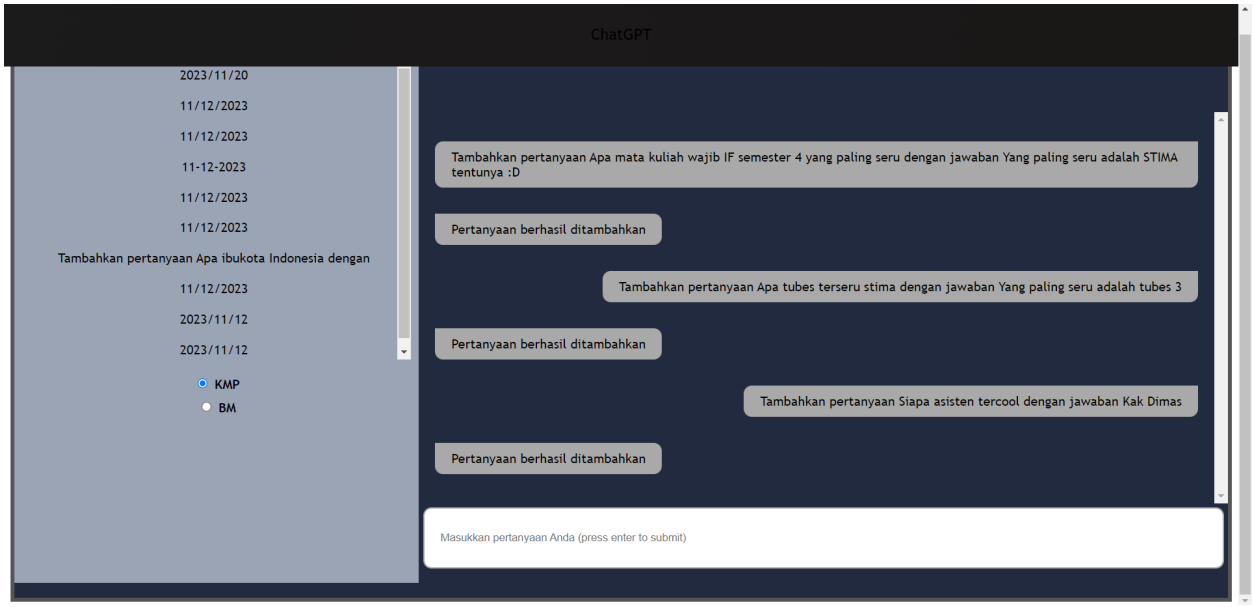
Gambar 4.3.1.1 Menambahkan pertanyaan ibukota Indonesia



Gambar 4.3.1.2 Menambahkan pertanyaan ibukota negara selain Indonesia yang berawalan huruf i

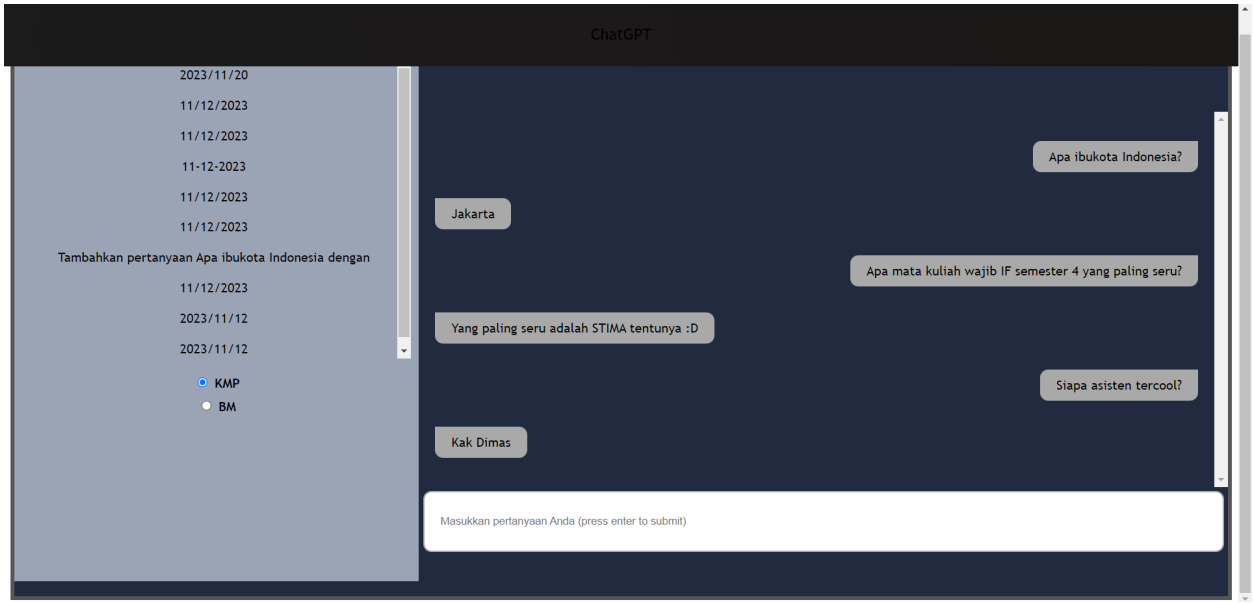


Gambar 4.3.1.3 Menambahkan pertanyaan dengan kata tanya kapan

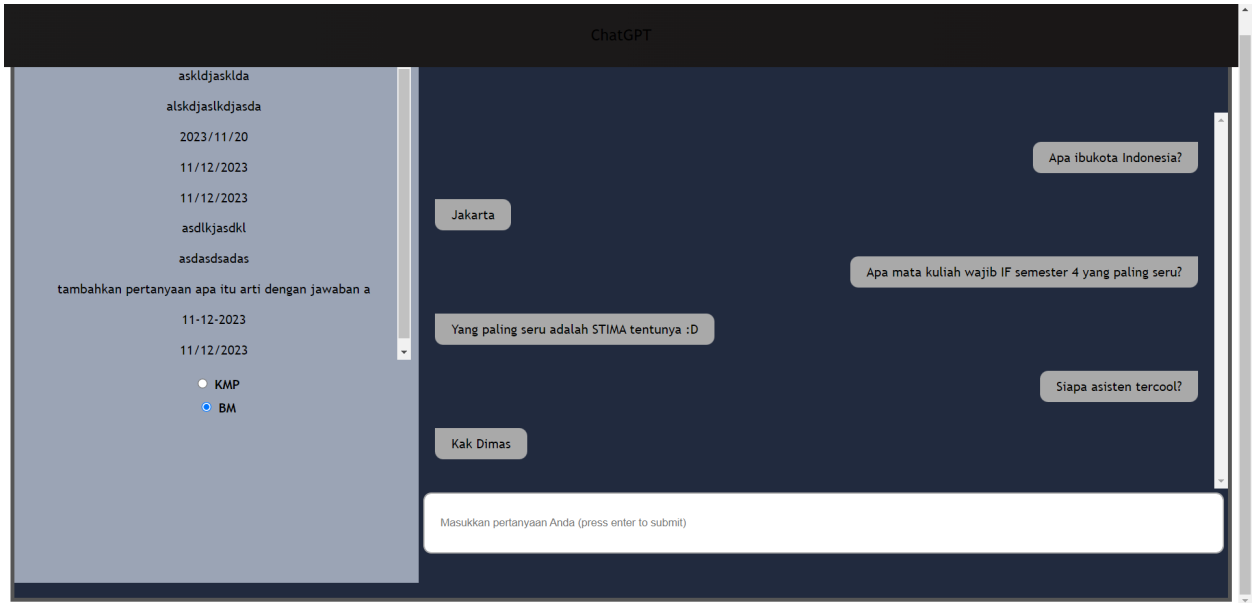


Gambar 4.3.1.4 Menambahkan pertanyaan wajib

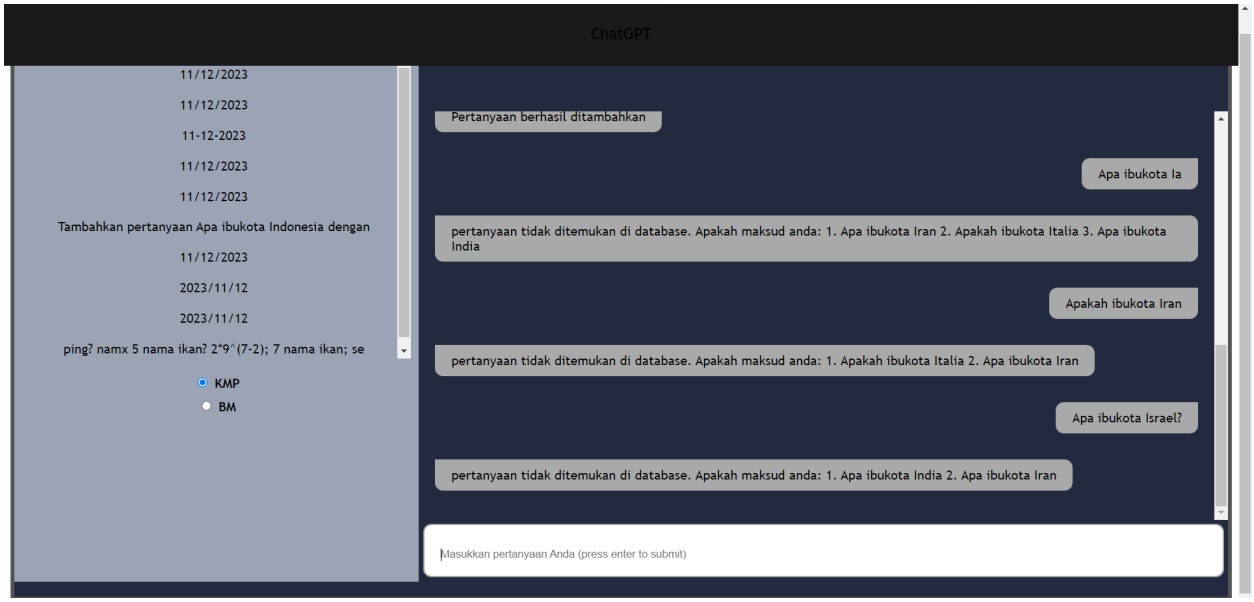
4.3.2 Fitur Pertanyaan Teks



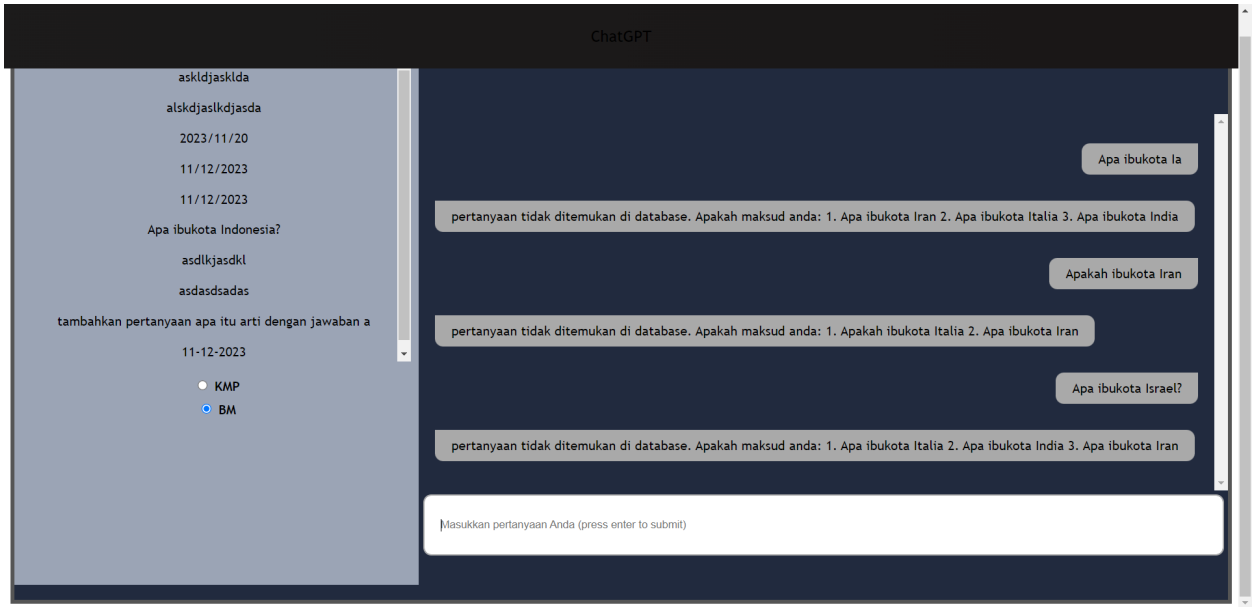
Gambar 4.3.2.1 Memberikan pertanyaan yang sama persis dengan pertanyaan pada basis data dengan pilihan algoritma KMP



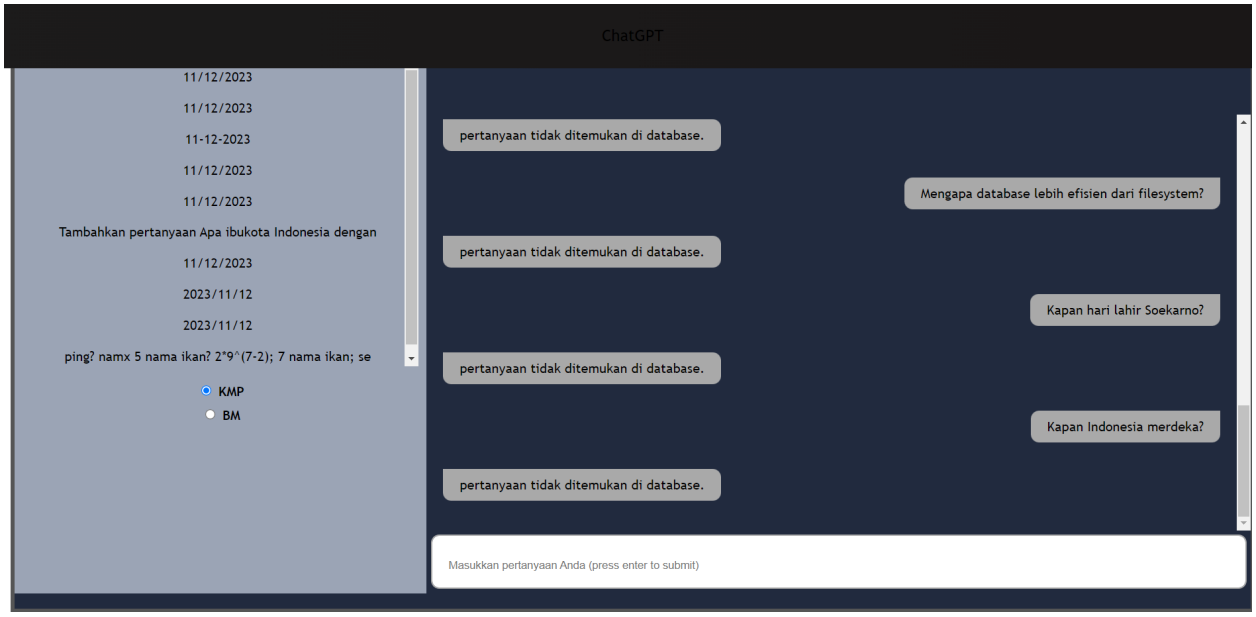
Gambar 4.3.2.2 Memberikan pertanyaan yang sama persis dengan pertanyaan pada basis data dengan pilihan algoritma BM



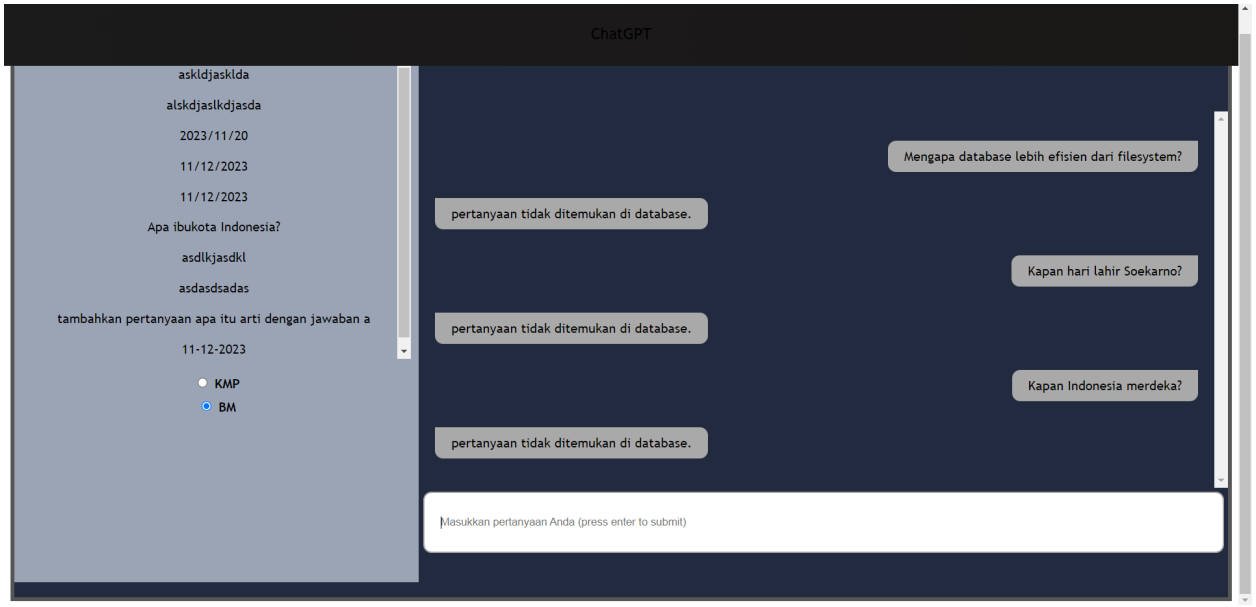
Gambar 4.3.2.3 Memberikan pertanyaan yang mirip dengan pertanyaan pada basis data dengan pilihan algoritma KMP



Gambar 4.3.2.4 Memberikan pertanyaan yang mirip dengan pertanyaan pada basis data dengan pilihan algoritma BM

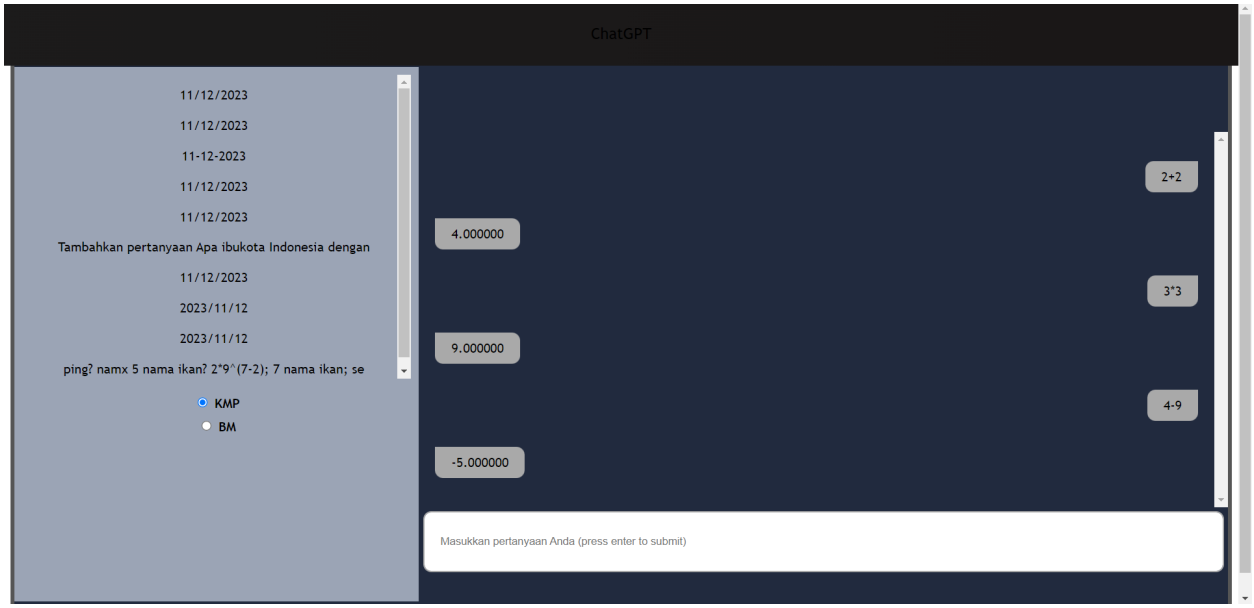


Gambar 4.3.2.5 Memberikan pertanyaan yang tidak mirip dengan pertanyaan pada basis data dengan pilihan algoritma KMP

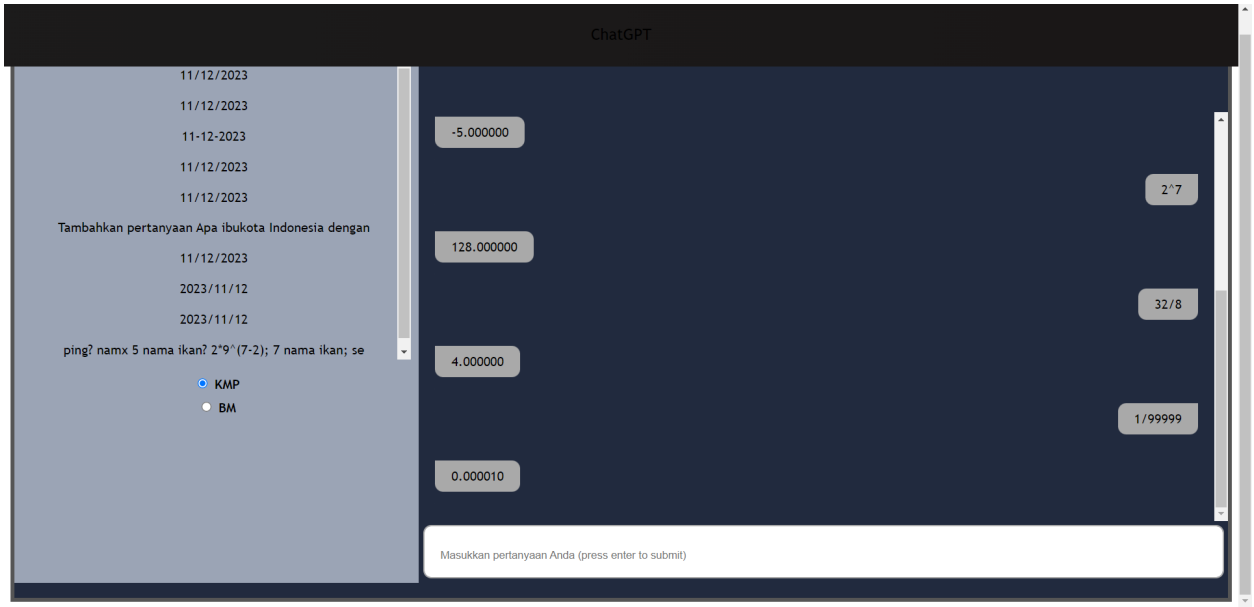


Gambar 4.3.2.6 Memberikan pertanyaan yang tidak mirip dengan pertanyaan pada basis data dengan pilihan algoritma BM

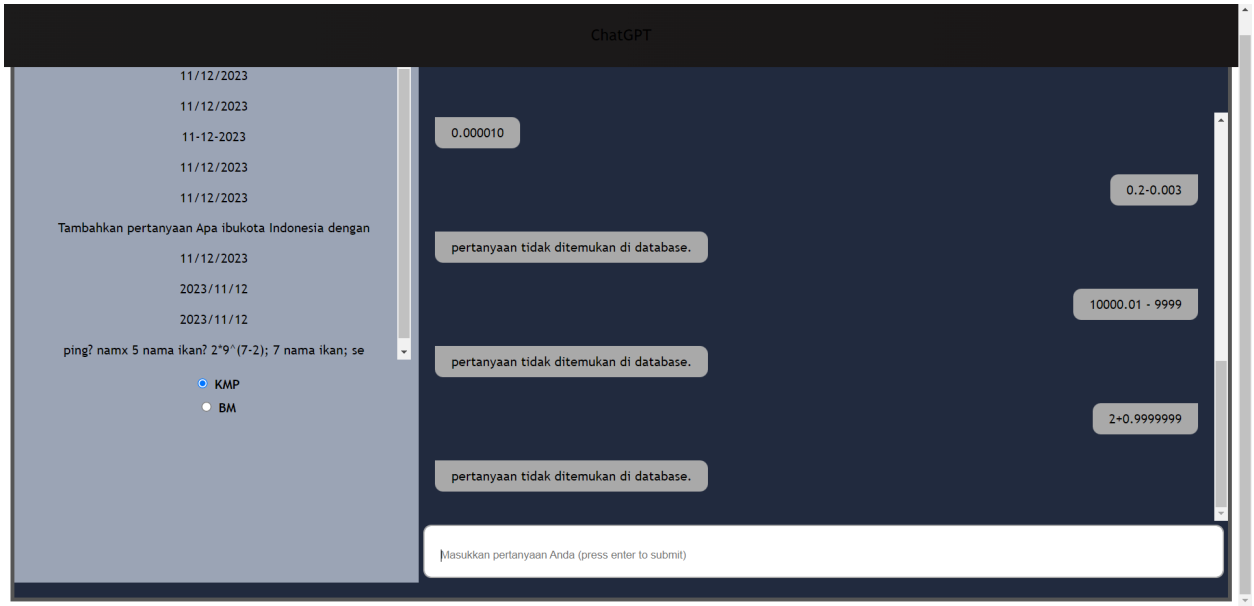
4.3.3 Fitur Kalkulator



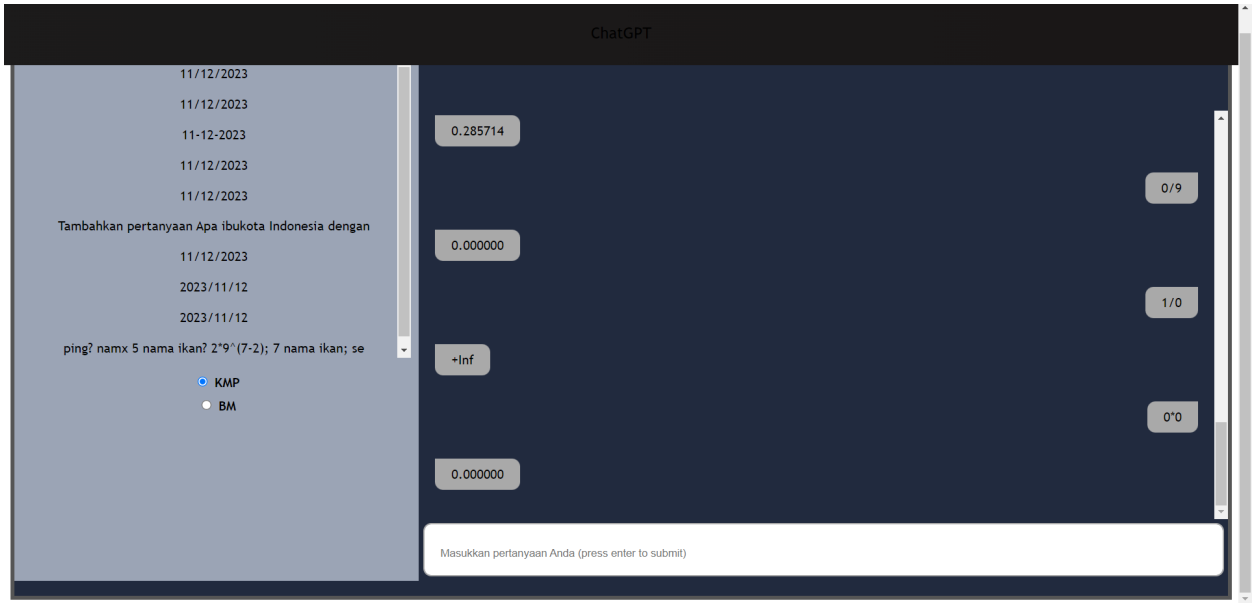
Gambar 4.3.3.1 Memberikan ekspresi penjumlahan, perkalian dan pengurangan



Gambar 4.3.3.2 Memberikan ekspresi perpangkatan dan pembagian

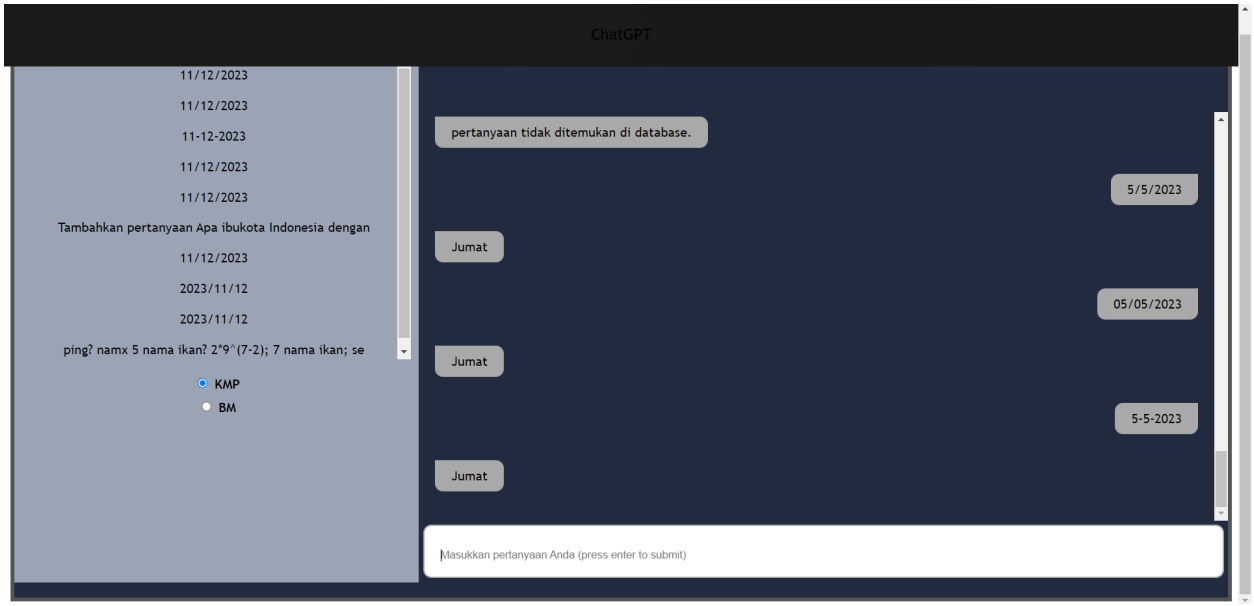


Gambar 4.3.3.3 Memberikan ekspresi dengan operan bilangan real

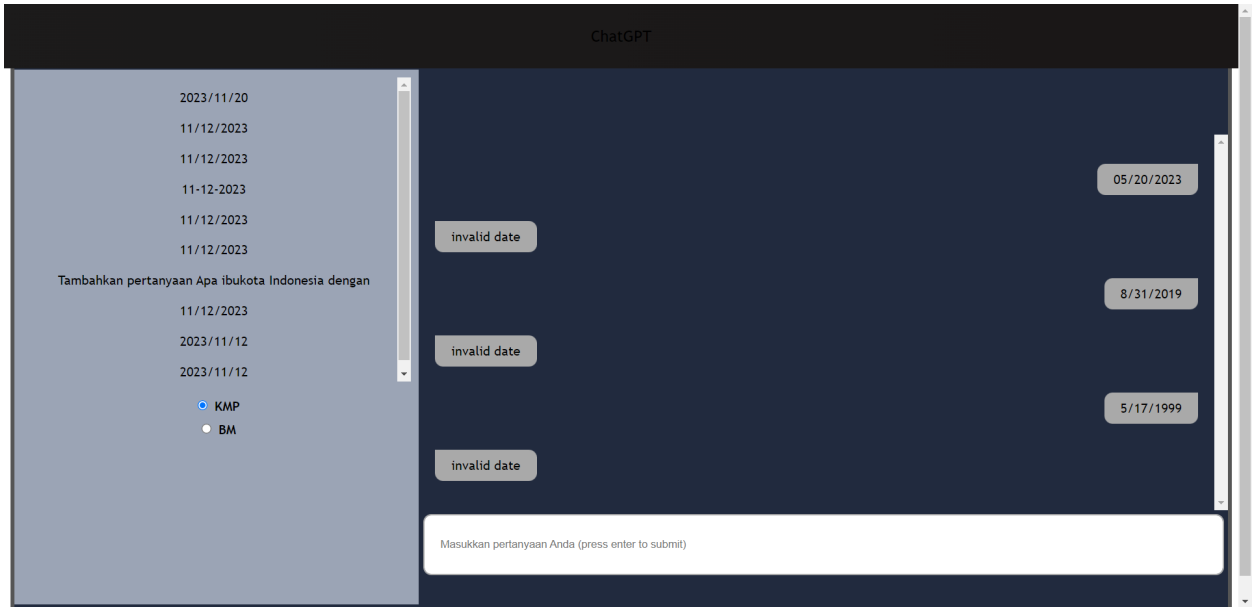


Gambar 4.3.3.4 Memberikan ekspresi yang melibatkan angka nol

4.3.4 Fitur Tanggal

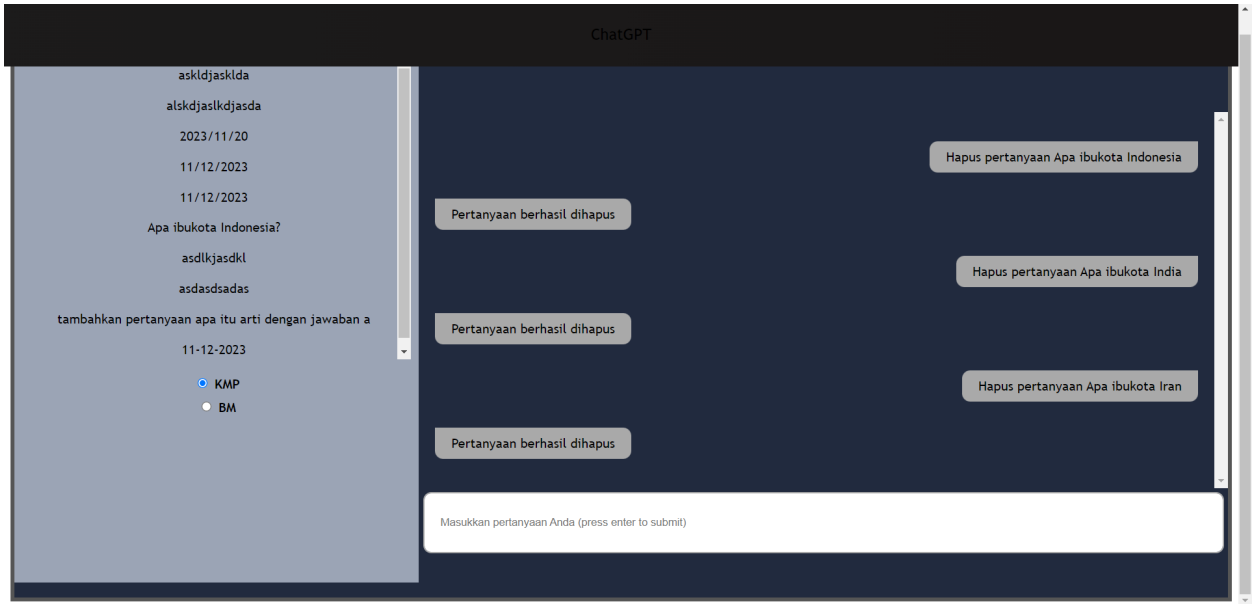


Gambar 4.3.4.1 Memberikan input tanggal yang dipisahkan dengan tanda hubung dan garis miring

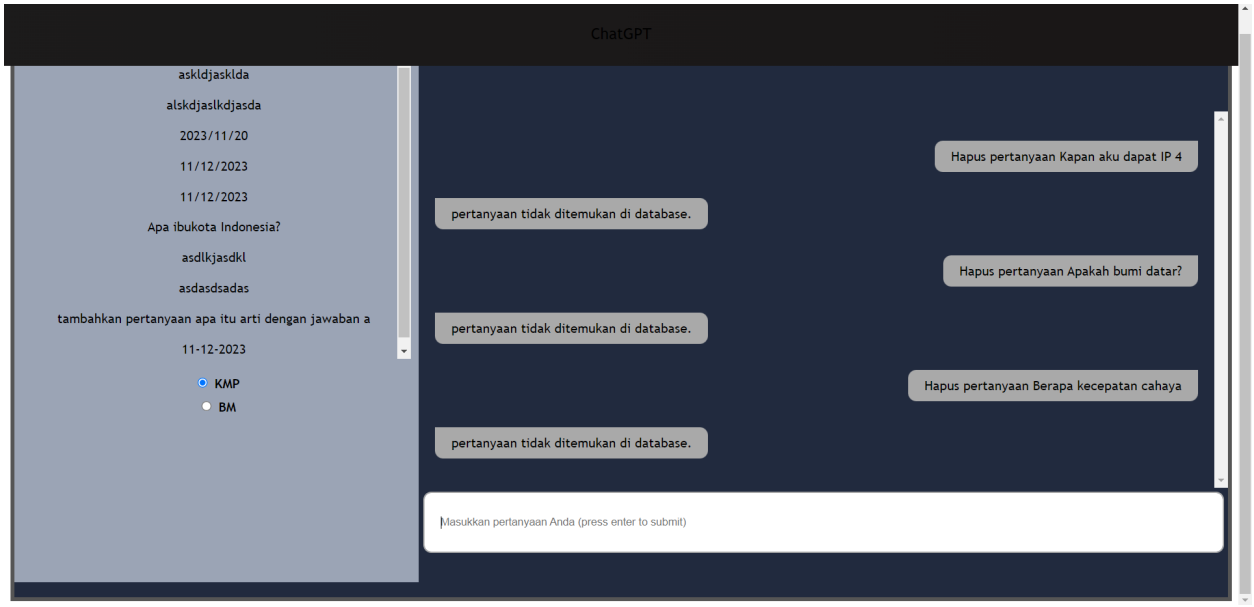


Gambar 4.3.4.2 Memberikan input tanggal dengan format dd/mm/yyyy

4.3.5 Menghapus Pertanyaan dari Database



Gambar 4.3.5.1 Menghapus pertanyaan yang ada di database



Gambar 4.3.5.2 Menghapus pertanyaan yang tidak ada di database

4.4. Analisis Hasil Pengujian

Secara umum, pencarian menggunakan algoritma KMP dan BM sudah memberikan hasil yang benar dan tepat. Hal ini dibuktikan pada tahap pengujian yang memberikan beberapa macam pertanyaan.

Pertama, pertanyaan yang persis sama dengan pertanyaan yang sudah ada di database. Untuk jenis pertanyaan ini, kedua jenis algoritma memberikan jawaban yang seluruhnya benar yang diperlihatkan pada Gambar 4.3.2.1 dan Gambar 4.3.2.2. Dengan demikian, dapat disimpulkan implementasi algoritma KMP dan BM untuk pencocokan string yang sama persis sudah benar.

Kedua, pertanyaan yang tidak sama persis dengan pertanyaan yang ada di database, tetapi memiliki tingkat kemiripan lebih dari atau sama dengan 90 persen. Program juga memberikan output yang benar untuk kasus ini yang dibuktikan pada Gambar 4.3.2.3 dan Gambar 4.2.3.4. Dapat disimpulkan, implementasi algoritma Levenshtein distance untuk menghitung tingkat kemiripan sudah benar.

Ketiga, pertanyaan yang tidak mirip dengan pertanyaan yang ada di database. Untuk kasus ini, program juga sudah memberikan output yang benar. Hal ini dibuktikan pada Gambar 4.3.2.5 dan Gambar 4.3.2.6. Dapat disimpulkan program sudah bisa menentukan pattern yang memiliki tingkat kecocokan kurang dari 90 persen dengan pertanyaan yang ada di database.

Fitur kalkulator dan tanggal sudah berfungsi dengan baik pada program. Hal ini dibuktikan dengan pengujian pada bagian 4.3.3 dan 4.3.4. Program dapat menerima ekspresi aritmatika penjumlahan, pengurangan, perkalian, pembagian, dan perpangkatan. Selain itu, program juga menolak ekspresi matematika yang tidak terdefinisi, seperti pembagian dengan nol. Pada fitur tanggal, chatbot sudah memberikan jawaban yang benar. Hal ini dibuktikan pada Gambar 4.3.4.1 dan Gambar 4.3.4.2. Selain itu, chatbot juga memberikan respon yang tepat berupa penolakan terhadap tanggal yang tidak dinyatakan dengan format dd/mm/yyyy atau dd-mm-yyyy.

Fitur lain yang menjadi salah satu fitur dasar pada program ini, yaitu menambah dan menghapus pertanyaan dari atau ke database juga sudah berfungsi dengan baik dan benar. Hal ini diperlihatkan pada pengujian 4.3.1 dan 4.3.5.

BAB 5
Kesimpulan dan Saran

5.1. Kesimpulan

Dari Tugas Besar 3 IF2211 Strategi Algoritma Semester II 2022/2023 ini kami menyimpulkan bahwa kami dapat memanfaatkan algoritma pencocokan string seperti KMP dan BM serta regular expression untuk menyelesaikan permasalahan aplikasi ChatGPT sederhana. Kedua algoritma tersebut sekilas mata terlihat mirip dengan *brute force*, namun kompleksitasnya lebih efisien yaitu O(m+n). Algoritma pencocokan string ini sangat bermanfaat untuk mengolah informasi berupa string untuk memenuhi kebutuhan sehari-hari.

5.2. Saran

Kami memiliki beberapa saran untuk disampaikan berkaitan dengan Tugas Besar 3 IF2211 Strategi Algoritma Semester II 2022/2023, yaitu:

- Aplikasi ini mungkin bisa diperluas implementasinya dengan bahasa-bahasa pemrograman lain yang lebih praktis digunakan.
- Aplikasi bisa ditambahkan halaman “welcome”.
- Aplikasi bisa ditambahkan fiturnya.

5.3. Refleksi

Dalam tugas besar ini kami menghadapi berbagai kendala. Pertama-tama, kami mengalami kesusahan ketika beradaptasi dengan bahasa yang jarang kami sentuh, seperti Javascript dan Golang. Lalu, halaman utama kami juga tidak serapi yang kami inginkan karena pengalaman kami dalam bidang front-end terbatas. Akhirnya, kami juga lumayan bingung dalam mengintegrasikan front-end dengan back-end secara efektif melalui file .json. Namun, berkat tugas besar ini, kami memperoleh berbagai pengetahuan baru

5.4. Tanggapan

Tugas besar ini sangat bermanfaat bagi kami karena kami dapat mengetahui penerapan algoritma pencocokan string dalam bentuk aplikasi berbasis web, sambil melakukan eksplorasi terhadap bahasa pemrograman Javascript dan Golang. Kami juga dapat melatih diri dalam melakukan integrasi front-end dan back-end.

Daftar Pustaka

- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf
- https://www.regexpal.com/
- https://www.programiz.com/javascript/regex

Lampiran

- Link repository: https://github.com/Breezy-DR/Tubes3_13521048
- Link YouTube: https://youtu.be/rWiu75sMwtU
- Link website: http://ec2-52-221-241-44.ap-southeast-1.compute.amazonaws.com:3000/