APLIKASI **CHATGPT** SEDERHANA DALAM BAHASA PEMROGRAMAN **NODE.JS** DENGAN ALGORITME ***STRING MATCHING*** DAN ***REGULAR EXPRESSION***

Disusun untuk memenuhi laporan tugas besar mata kuliah IF2211 Strategi Algoritma semester 4 di Institut Teknologi Bandung.

Disusun oleh kelompok KitKat:

Ryan Samuel Chandra (13521140)

Sulthan Dzaky Alfaro (13521159)

Satria Octavianus Nababan (13521168)

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

PRAKATA

Manusia diciptakan oleh Tuhan Yang Maha Esa, dengan dilengkapi oleh akal budi. Meskipun demikian, kita tidak pernah bisa melanjutkan hidup tanpa berkat dan rahmat-Nya. Maka dari itu, pertama-tama, kami panjatkan puji dan syukur kepada Tuhan, sebab akhirnya kami dapat menyelesaikan laporan ini tepat waktu. Kami juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T., juga Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., serta para asisten mata kuliah IF2211 Strategi Algoritma, yang telah membagikan ilmu serta memberikan kesempatan bagi kami untuk belajar lebih dalam di dunia informatika.

Berikut disajikan laporan lengkap dengan judul "Aplikasi ChatGPT sederhana dalam Bahasa Pemrograman Node.js dengan Algoritme *String Matching* dan *Regular Expression*". Laporan ini selain dibuat agar membawa manfaat bagi masyarakat, juga secara khusus disusun untuk memenuhi tugas besar terakhir pada mata kuliah IF2211 Strategi Algoritma di semester 4 Teknik Informatika Institut Teknologi Bandung.

Dengan kerja sama serta pertemuan rutin selama dua-tiga minggu terakhir ini, ditambah pengumpulan informasi dan uji coba program, kami berusaha sebaik mungkin untuk memenuhi tujuan pembuatan yang telah ditetapkan, meskipun harus bekerja di tengah-tengah tekanan waktu dan hambatan dari lingkungan.

Kami berharap percobaan yang telah kami lakukan dapat menjadi sesuatu yang memuaskan bagi semua pembaca. Tetapi, kami pun menyadari bahwa masih banyak kekurangan yang bisa diperbaiki. Maka dari itu, kami mohon kritik dan saran yang membangun untuk perkembangan program kami. Terima kasih, selamat membaca.

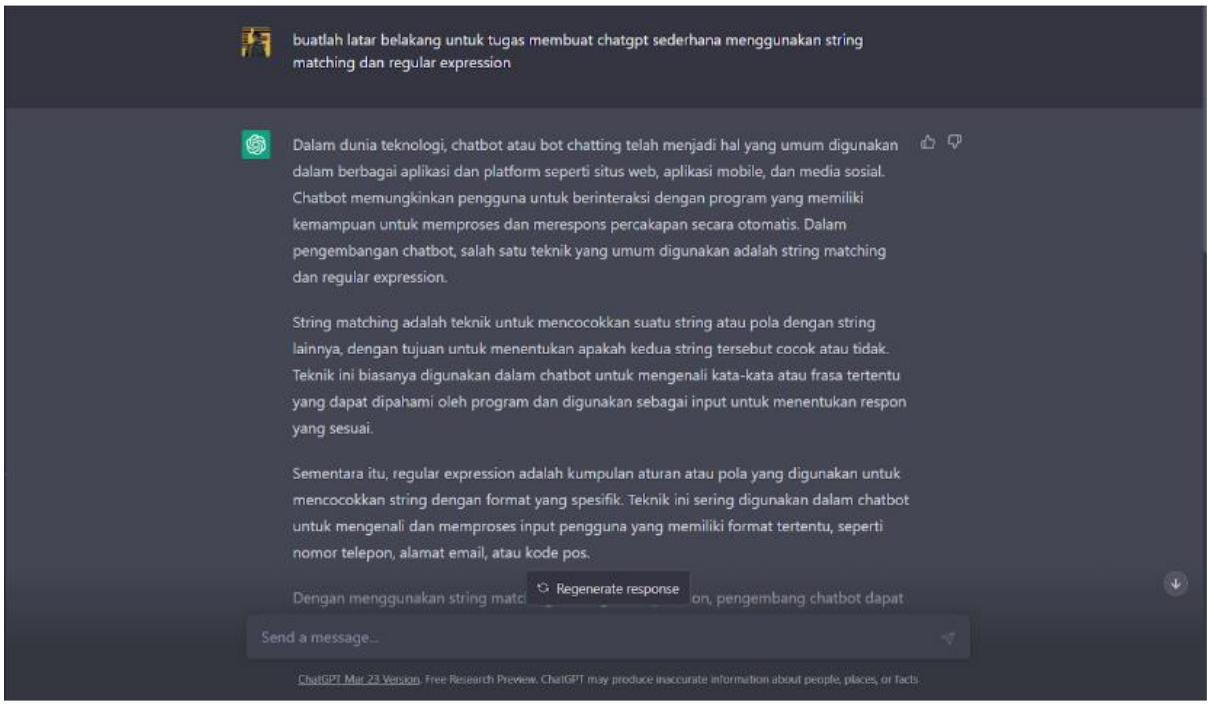
Bandung, 5 Mei 2023

Penyusun Laporan

Daftar Isi

BAB 1
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 Tampilan *Chatbot* ChatGPT

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 respons 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.

Deskripsi tugas:

Dalam tugas besar ini, mahasiswa 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 dibuat 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 algoritme **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.

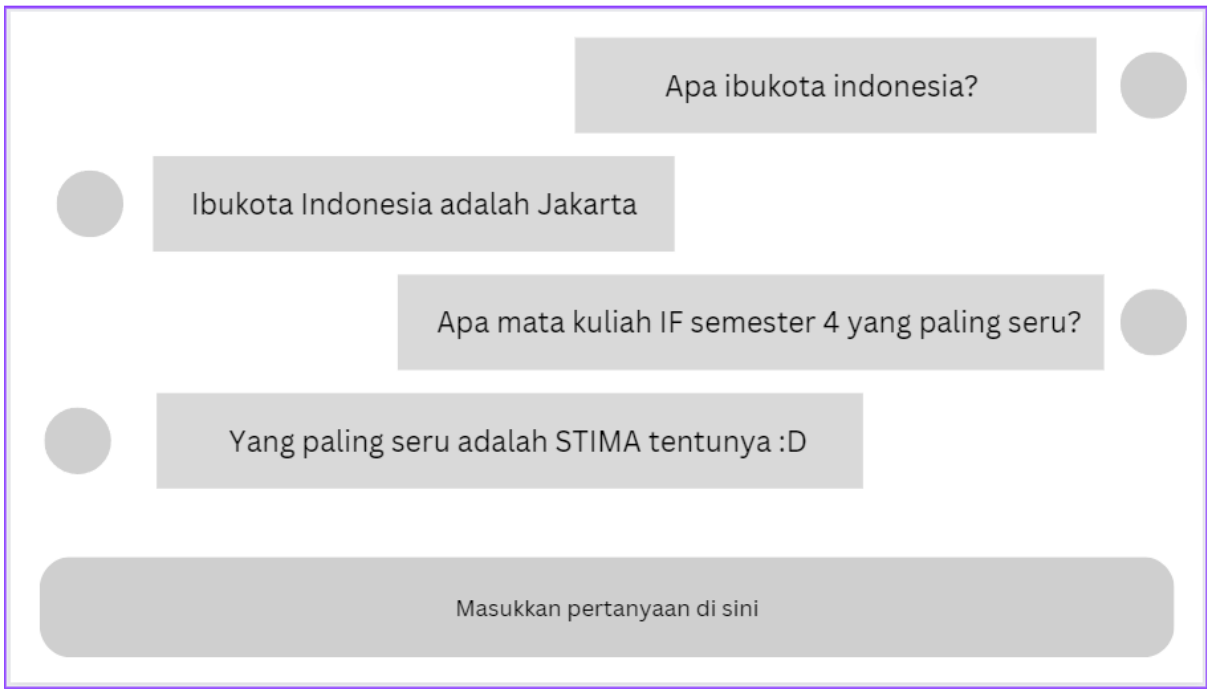
4. Tambah pertanyaan dan jawaban ke *database*

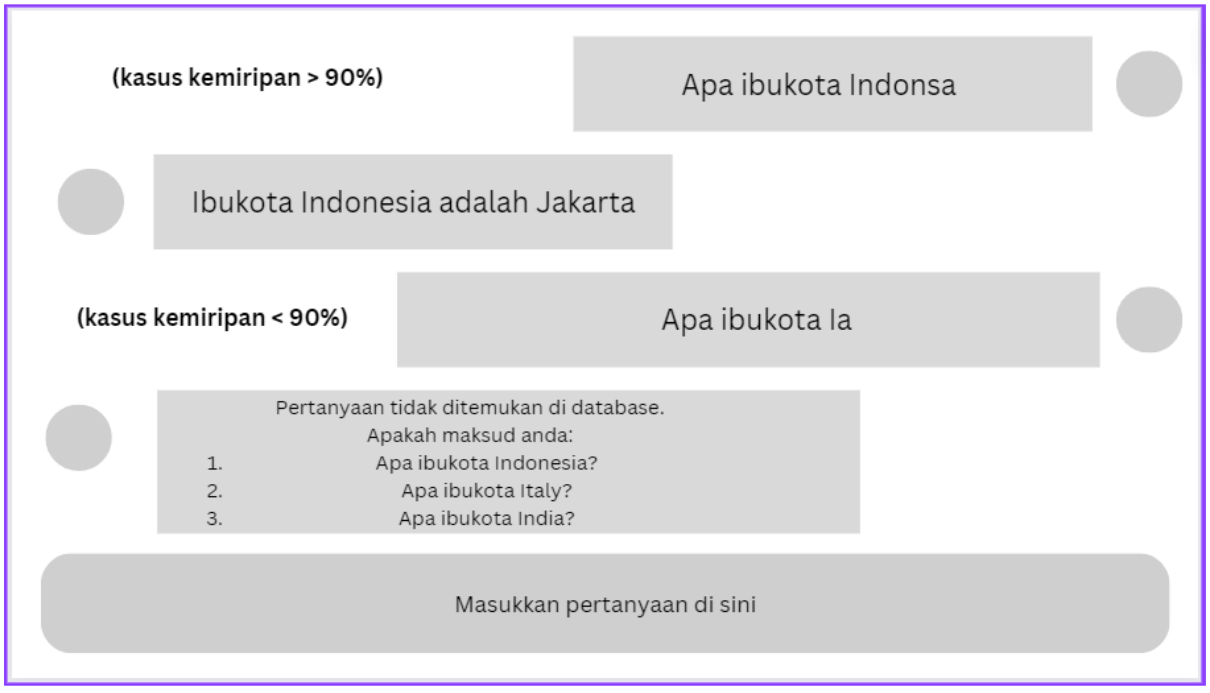
Pengguna dapat menambahkan pertanyaan dan jawabannya sendiri ke *database* dengan *query* contoh “Tambahkan pertanyan xxx dengan jawaban yyy”. Menggunakan algoritme *string matching* untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbaharui.

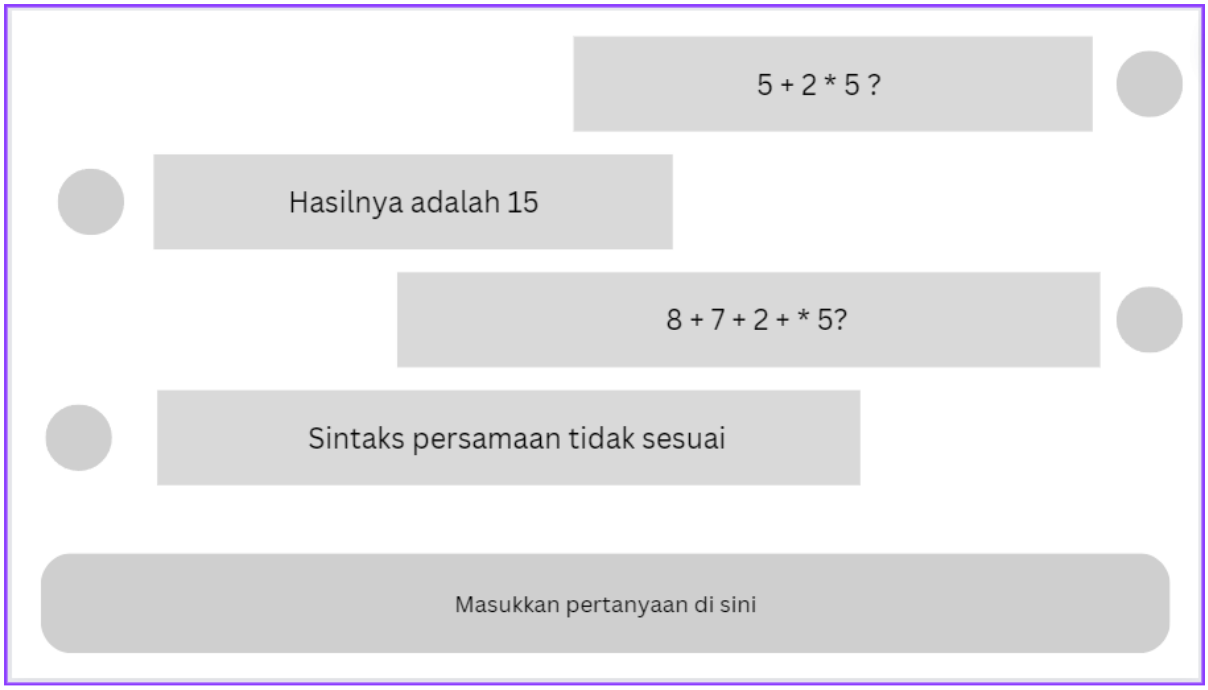
5. Hapus pertanyaan dari *database*

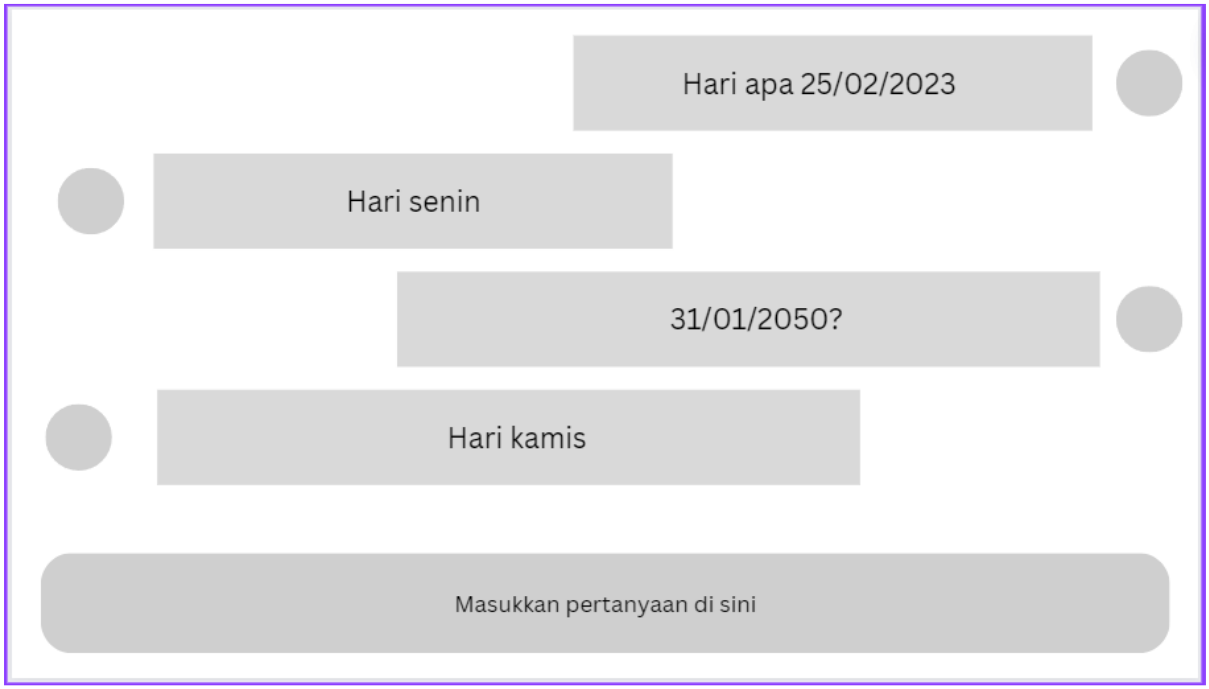
Pengguna dapat menghapus sebuah pertanyaan dari *database* dengan *query* contoh “Hapus pertanyaan xxx”. Menggunakan algoritme *string matching* untuk mencari pertanyaan xxx tersebut pada *database*.

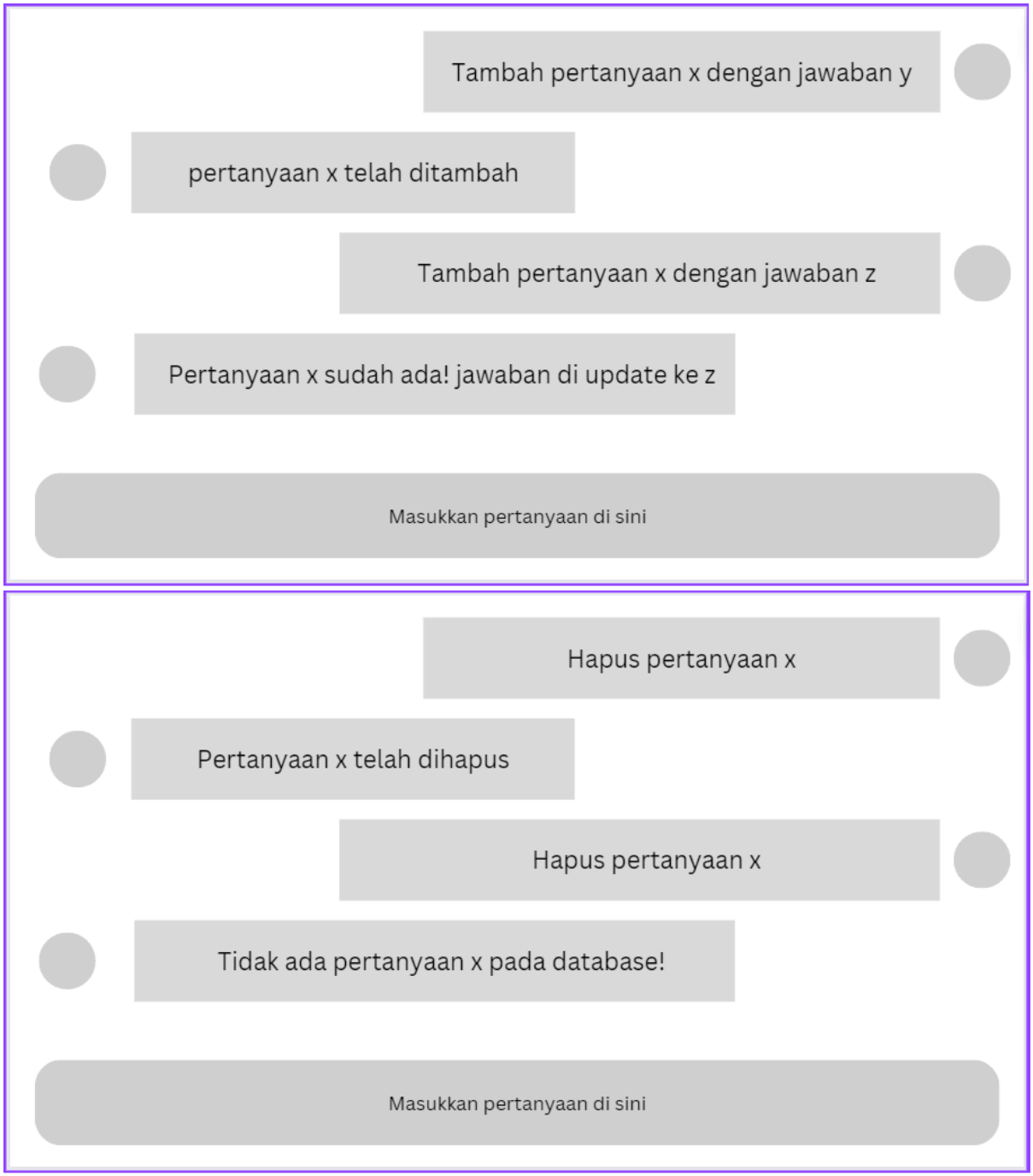
Klasifikasi dilakukan menggunakan ***regex*** dan terklasifikasi layaknya bahasa sehari-hari. Algoritma *string matching* KMP dan BM digunakan untuk klasifikasi *query* teks. Tersedia *toggle* untuk memilih algoritme KMP atau BM. Semua pemrosesan respons dilakukan pada sisi ***backend***. Jika ada pertanyaan 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**)**






5 + 2 * 5 ?
Hasilnya adalah 15
8 + 7 + 2 + * 5?
Sintaks persamaan tidak sesuai
Masukkan pertanyaan di sini


Hari apa 25/02/2023
Hari senin
31/01/2050?
Hari kamis
Masukkan pertanyaan di sini



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* di-*click*, maka akan me-*restore* seluruh pertanyaan dan jawaban di halaman utama.

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").

(Sumber: Spesifikasi Tugas Besar 3 IF2211 Strategi Algoritma Semester 2 Tahun 2022/2023, diakases dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tubes3-Stima-2023.pdf pada tanggal 4 Mei 2023)

BAB 2
LANDASAN TEORI

2.1 *Regular Expression*

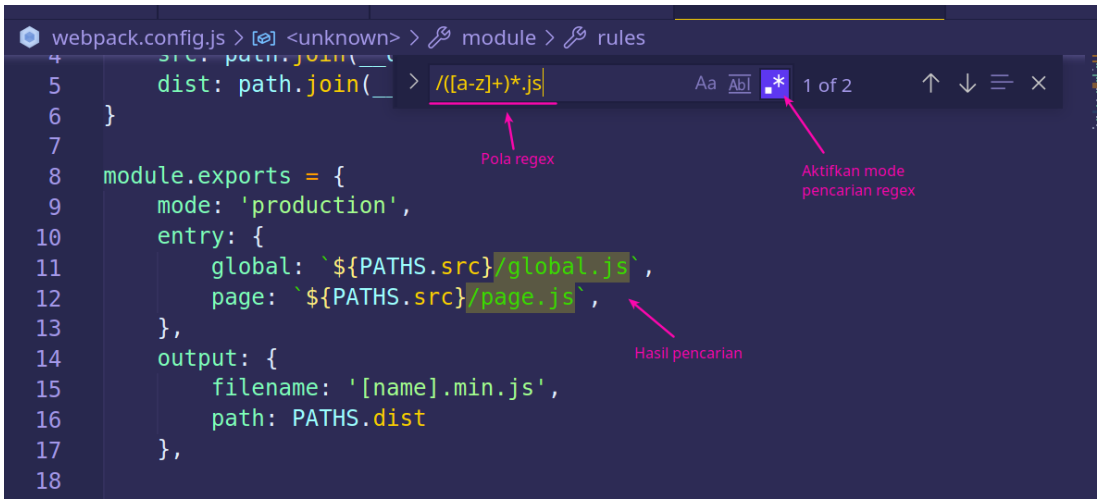
Regular Expression (*RegEx*) merupakan sebuah teks (*string*) yang mendefinisikan sebuah pola pencarian sehingga dapat membantu kita untuk melakukan *matching* (pencocokan), *locate* (pencarian), dan manipulasi teks. Konsep tentang regex pertamakali muncul di tahun 1951, ketika seorang ilmuwan / matematikawan bernama Stephen Cole Kleene memformulasikan definisi tentang bahasa formal [3].

Berikut ini beberapa contoh pemanfaatan *RegEx* dalam pemrograman:

1. Validasi data seperti *password* masukan pengguna. Program utama dibuat untuk memeriksa apakah nilai dari *field password* sudah sesuai atau tidak dengan yang ada pada atribut berisi *RegEx*, misalnya bernama "pattern".

```
<input type="password" name="password" pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}" />
```

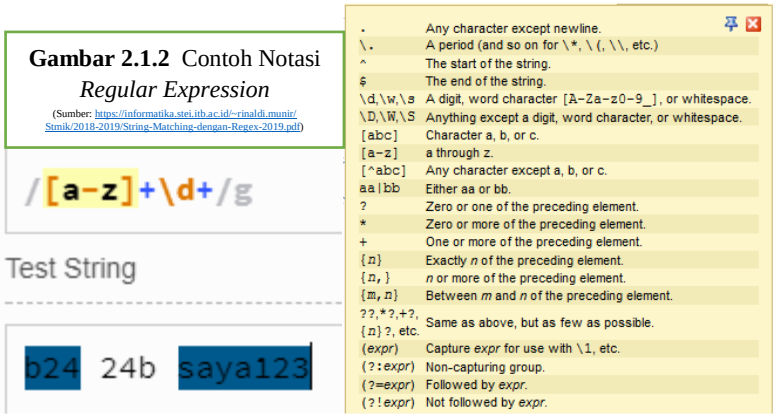
2. Mencari sebuah pola pada *text editor*. Misalnya, ingin dicari di mana saja penulisan nama *file* yang berakhiran *.js* dalam Visual Studio Code. Hal tersebut bahkan dapat digunakan pada saat *find* dan *replace*.



Gambar 2.1.1 Contoh Penerapan *Regular Expression* dalam *Text Editor*
(Sumber: https://www.petanikode.com/regex/)

Salah satu bagian tersulit dalam mempelajari *regular expression* adalah membuat sendiri dan menghafal simbol-simbol yang dapat dipakai. Berikut adalah notasi umum *RegEx* beserta contohnya dalam pencarian pola:

Gambar 2.1.2 Contoh Notasi *Regular Expression*
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf)



2.2 *Pattern Matching* dengan Algoritme *Brute Force*

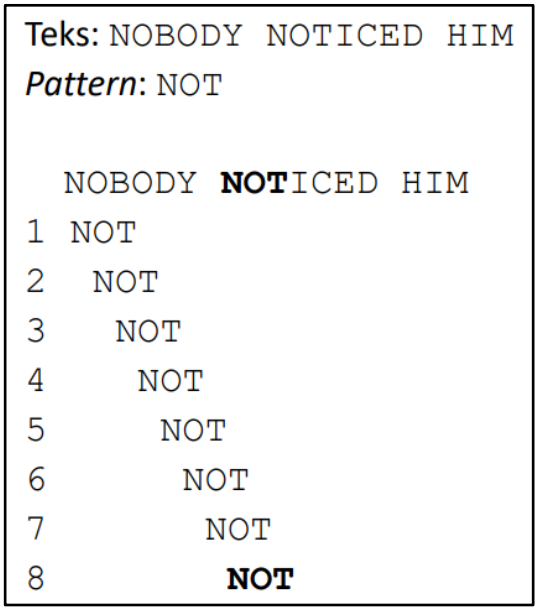
Misalnya, diberikan sebuah teks *T*, yaitu *string* dengan panjang *n* karakter. Lalu, diberikan pula *pattern P*, yaitu *string* dengan panjang *m* karakter, dengan asumsi *m* jauh lebih kecil daripada *n*. *Pattern matching* akan mengembalikan indeks lokasi pertama di dalam teks, yang bersesuaian dengan *pattern*. Sebagai contoh:

T: `the rain in spain stays mainly on the plain`

P: `main`

Aplikasi *pattern matching* sangat beragam, mulai dari pencarian dalam *text editor*, *web search engine* seperti Google, analisis citra, bahkan dalam bidang *bioinformatics*.

Dengan algoritme *brute force*, *string matching* dilakukan dengan memeriksa setiap posisi dalam teks, untuk memastikan *pattern* mulai dari posisi tertentu. Contohya sebagai berikut:



Gambar 2.2.1 Contoh *String Matching* dengan *Brute Force*

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

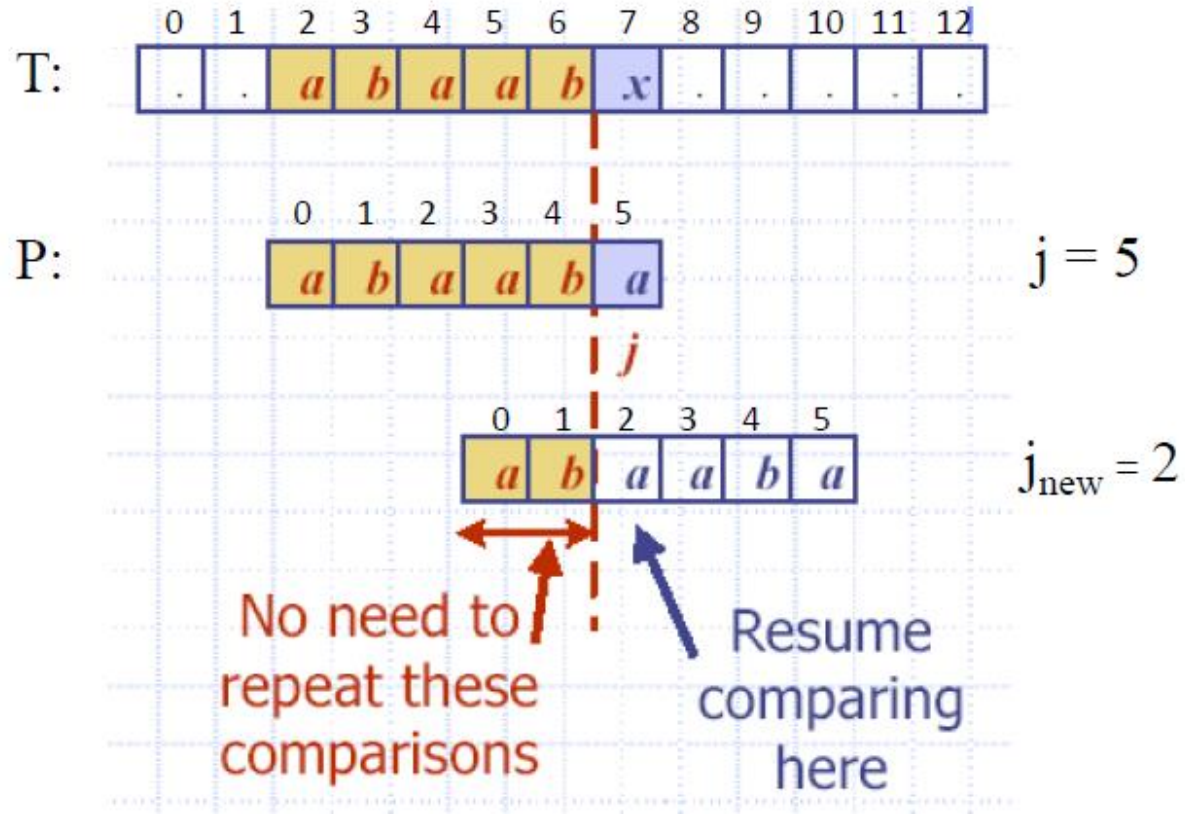
Dalam kasus terburuk, terdapat $m(n-m+1)$ jumlah perbandingan, sehingga menghasilkan kompleksitas $O(mn)$. Sedangkan kasus terbaik secara umum terjadi saat karakter pertama pada P tidak pernah sama dengan karakter pertama di posisi mana pun pada T, menghasilkan kompleksitas $O(n)$. Sedangkan, kasus rata-ratanya memiliki kompleksitas $O(m+n)$.

Algoritme *brute force* untuk *string matching* akan lebih cepat jika jangkauan alfabet pada teks cukup luas (misalnya 0-9, A-Z, a-z, dst.), namun lebih lambat jika sebaliknya (misalnya untuk teks biner).

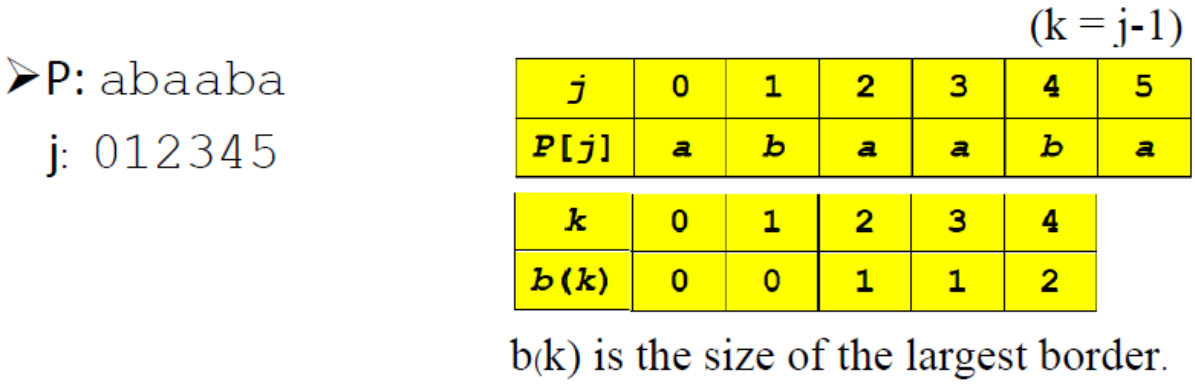
2.3 Algoritme Knuth-Morris-Pratt (KMP)

Algoritme Knuth-Morris Pratt merupakan algoritme pencocokan *substring* / pencarian *string* dari kumpulan beberapa *string*. Algoritme ini hampir mirip dengan *brute force*, namun dalam pencocokan *string*-nya, apabila ada ketidakcocokan, algoritme akan lompat sesuai dengan *border function*-nya. *Border function* itu sendiri merupakan nilai maksimum banyaknya huruf *prefix* dan *suffix* yang sama pada *pattern* yang ingin dicek.

Misalnya, sebuah *mismatch* terjadi sehingga $T[i] \neq P[j]$, maka *pattern* harus digeser ke kanan sebanyak jumlah karakter terbanyak pada *prefix* ($P[0..j-1]$) yang juga merupakan *suffix* ($P[1..j-1]$). Dengan demikian, proses perbandingan yang "tidak berguna" dapat dihindari.



Gambar 2.3.1 Contoh *String Matching* dengan Algoritme KMP
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)



j	0	1	2	3	4	5
P[j]	a	b	a	a	b	a

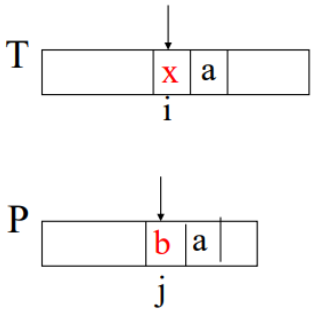
k	0	1	2	3	4
b(k)	0	0	1	1	2

Gambar 2.3.1 Contoh Tabel Hasil Komputasi *Border Function* untuk "abaaba"
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

2.4 Algoritme Boyer-Moore (BM)

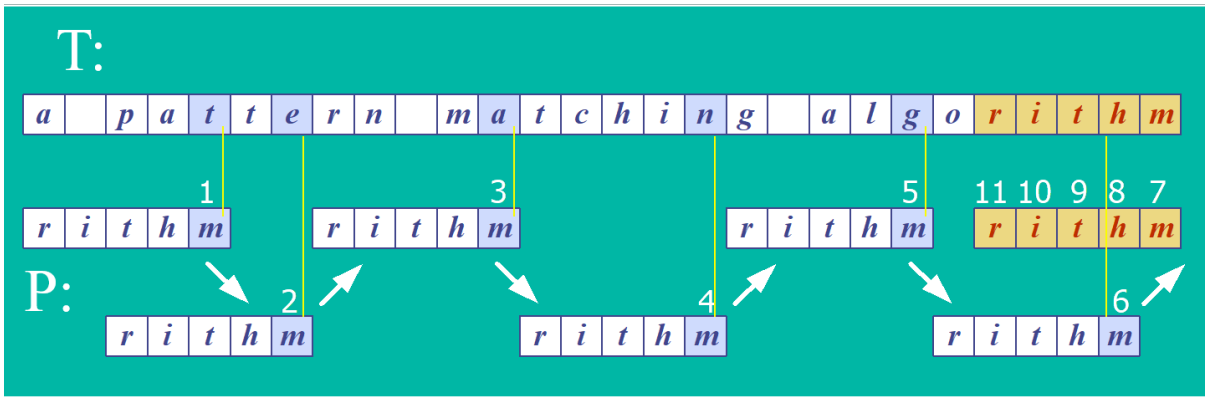
Algoritme Boyer-Moore adalah algoritme *pattern matching* berdasarkan dua buah teknik. Pertama, *looking-glass technique*, yaitu mencari *pattern* pada teks dengan cara bergerak mundur sepanjang *pattern* (mulai dari karakter terakhir). Kedua, *character-jump technique*.

Character-jump technique dilakukan setiap kali terjadi *mismatch* (ketidakcocokan) antara $T[i] = x$ dengan $P[j]$. Ada setidaknya tiga kemungkinan yang perlu diperhatikan secara berurutan dan ditangani dengan cara yang berbeda:



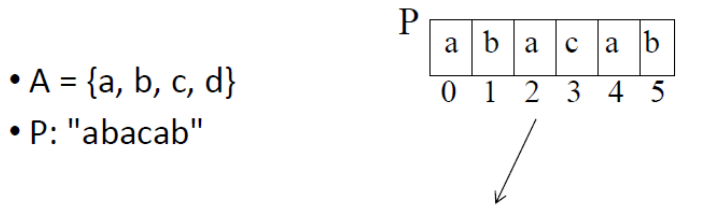
Gambar 2.2.2 Contoh *Mismatch* pada Algoritme Boyer-Moore
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

1. Jika P mengandung x di suatu tempat dengan indeks lebih kecil dari j, maka P digeser ke kanan untuk menyesuaikan posisi kemunculan terakhir x pada P dengan $T[i]$.
2. Jika P mengandung x di suatu tempat dengan indeks lebih besar dari j, maka P digeser ke kanan sebanyak satu karakter saja ($P[j]$ bersesuaian dengan $T[i+1]$).
3. Jika kedua kasus di atas tidak terpenuhi, maka P digeser ke kanan sehingga $P[0]$ bersesuaian dengan $T[j+1]$.



Gambar 2.2.3 Contoh *String Matching* dengan Algoritme Boyer-Moore
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

Untuk mendukung keberjalanan *string matching* dengan algoritme ini, dibutuhkan *last occurrence function*, $L(x)$, yang terdefinisi sebagai indeks i terbesar yang memenuhi $P[i] = x$, atau -1 jika tidak ada nilai i yang sesuai.



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

Gambar 2.2.4 Contoh Tabel Hasil Komputasi *L*(*x*)
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)

2.5 *Levenshtein Distance* dan Persentase Kemiripan

Algoritme *Levenshtein Distance*, atau yang sering disebut sebagai *Edit Distance*, merupakan algoritme untuk mencari jumlah perbedaan antara dua buah *string*. Algoritme ini ditemukan pada tahun 1965 oleh seorang ilmuwan Rusia bernama Vladimir Levenshtein [4].

Pada dasarnya, algoritme tersebut akan menghitung jumlah minimum dari upaya transformasi suatu *string* menjadi *string* lain. Transformasi ini meliputi penggantian, penghapusan, dan penyisipan. Proses ini juga digunakan untuk mengoptimalkan pencarian *string* karena apabila dilakukan pencarian setiap kombinasi operasi-operasi string tersebut maka akan membutuhkan sumber daya yang besar dan tidak efektif [4].

Levenshtein distance melibatkan matriks dua dimensi. Matriks tersebut akan berisi nilai berupa jumlah operasi penghapusan, penyisipan dan penukaran yang dibutuhkan dalam mengubah *string* sumber ke *string* target [4].

Langkah-langkah yang perlu dilakukan untuk mendapatkan *edit distance* adalah:

1. Misalkan *S* = *string* sumber, dan *T* = *string* target
2. Langkah 1: Inisialisasi
   a) Hitung panjang *S* dan *T*, misalkan *m* dan *n*
   b) Buat matriks berukuran [0..*m*] baris dan [0..*n*] kolom
   c) Inisialisasi baris pertama dengan 0..*n*
   d) Inisialisasi kolom pertama dengan 0..*m*
3. Langkah 2: Proses
   a) Periksa *S*[*i*] untuk 1 < *i* < *n*
   b) Periksa *T*[*j*] untuk 1 < *j* < *m*
   c) Jika *S*[*i*] = *T*[*j*], maka entrinya adalah nilai yang terletak pada tepat di diagonal atas sebelah kiri, yaitu *d*[*i*, *j*] = *d*[*i*-1, *j*-1]
   d) Jika *S*[*i*] ≠ *T*[*j*], maka entrinya adalah *d*[*i*, *j*] minimum dari:
      - Nilai yang terletak tepat di atasnya, ditambah satu, yaitu *d*[*i*, *j*-1]+1
      - Nilai yang terletak tepat di kirinya, ditambah satu, yaitu *d*[*i*-1, *j*]+1
      - Nilai yang terletak pada tepat di diagonal atas sebelah kirinya, ditambah satu, yaitu *d*[*i*-1, *j*-1]+1

Setelah didapatkan *distance* (*d*) dari kedua *string*, maka dapat ditentukan formula untuk menghitung *similarity percentage* dari kedua *string*: $SP = \frac{\max(\text{length}(S),\ \text{length}(T)) - d}{d}$

2.6 Node.js dan MySQL

Node.js adalah lingkungan *runtime* JavaScript yang *open-source* dan lintas *platform*. Node.js memungkinkan *developer* untuk membuat aplikasi *front-end* dan *back-end* menggunakan JavaScript. Ini dirilis pada tahun 2009 oleh Ryan Dahl.

Gambar 2.6.1 Logo Node.js dan MySQL
(Sumber: Wikipedia)

Open-source berarti kode sumber untuk Node.js tersedia untuk umum, dan dapat dikelola oleh kontributor dari seluruh dunia. Lintas *platform* berarti tidak bergantung pada perangkat lunak sistem operasi apa pun. Node.js dapat bekerja di Linux, macOS, maupun Windows.

Ketika kode JavaScript ditulis di editor teks, kode tersebut tidak dapat melakukan tugas sama sekali kecuali dijalankan dalam lingkungan *runtime*. Peramban seperti Chrome dan Firefox memiliki lingkungan *runtime*-nya sendiri. Itu sebabnya mereka dapat menjalankan kode JavaScript. Sebelum Node.js dibuat, JavaScript hanya bisa berjalan di *browser*, sehingga hanya digunakan untuk membangun aplikasi *front-end*. Node.js pun menyediakan lingkungan *runtime* di luar *browser*. Ini memungkinkan seseorang untuk membangun aplikasi *back-end* dengan menggunakan bahasa pemrograman JavaScript.

MySQL adalah sistem manajemen basis data relasional (RDBMS / *Relational Database Management System*) yang *open-source* dengan model klien-server. RDBMS adalah perangkat lunak atau layanan yang digunakan untuk membuat dan mengelola basis data berdasarkan model relasional. Klien-server berarti komputer yang menginstal dan menjalankan perangkat lunak RDBMS disebut klien. Setiap kali mereka perlu mengakses data, mereka terhubung ke server RDBMS.

Database sendiri hanyalah kumpulan data terstruktur. *Database* adalah tempat di mana data disimpan dan diatur. Kata "relasional" berarti bahwa data yang disimpan dalam kumpulan data diatur sebagai tabel. Setiap tabel berhubungan dalam beberapa hal. Jika perangkat lunak tidak mendukung model data relasional, maka disebut DBMS saja.

Beberapa aplikasi web besar seperti Facebook, Twitter, YouTube, Google, dan Yahoo! semua menggunakan MySQL untuk keperluan penyimpanan data. Meskipun awalnya dibuat untuk penggunaan terbatas, sekarang ia kompatibel dengan banyak platform komputasi penting seperti Linux, macOS, Microsoft Windows, dan Ubuntu.

Query pada SQL dapat menginstruksikan server untuk melakukan operasi tertentu:

a) **Permintaan data**: meminta informasi spesifik dari *database* yang ada.
b) **Manipulasi data**: menambah, menghapus, mengubah, menyortir, dan operasi lain untuk mengubah data, nilai, atau visual.

c) **Identitas data**: menentukan tipe data, misalnya mengubah data numerik menjadi bilangan bulat. Ini juga termasuk mendefinisikan skema atau hubungan setiap tabel dalam *database*.
d) **Kontrol akses data**: menyediakan teknik keamanan untuk melindungi data, ini termasuk memutuskan siapa yang dapat melihat atau menggunakan informasi apa pun yang disimpan dalam *database*.

MySQL sangat terkenal dan banyak dipakai karena fleksibel dan mudah digunakan, memiliki performa dan kecepatan yang tinggi, memenuhi standar industri, dan aman.

2.7 ChatGPT

ChatGPT adalah alat pemrosesan bahasa alami (*natural language*) yang digerakkan oleh teknologi AI, yang memungkinkan seseorang melakukan percakapan layaknya percakapan sehari-hari dengan sebuah *chatbot*. Ia dapat menjawab pertanyaan dan membantu orang-orang menyelesaikan tugas, seperti menulis email, esai, bahkan kode program.

Penggunaan saat ini terbuka untuk umum secara gratis karena ChatGPT sedang dalam tahap penelitian dan pengumpulan *feedback*. ChatGPT dibuat oleh OpenAI, sebuah perusahaan AI (*Artificial Intelligence* / kecerdasan buatan) dan penelitian. Perusahaan tersebut meluncurkan ChatGPT pada 30 November 2022.

Jika mesin pencari mengindeks halaman web di internet untuk membantu pengguna menemukan informasi yang mereka minta, ChatGPT tidak memiliki kemampuan itu, melainkan menggunakan informasi yang dipelajari dari data untuk menghasilkan respons. Meski terlihat sangat impresif, ChatGPT tetap memiliki keterbatasan, misalnya ketidakmampuan untuk menjawab pertanyaan yang disusun dengan cara tertentu, serta kurangnya kualitas tanggapan yang diberikannya.

Meskipun beberapa orang menggunakan ChatGPT untuk melakukan berbagai pekerjaan rumit, seperti menulis kode atau mendeteksi *malware*, ChatGPT dapat digunakan untuk memulai percakapan biasa. Beberapa pembuka percakapan bisa sesederhana, "Saya lapar, makanan apa yang harus saya beli?" atau yang rumit seperti, "Menurut Anda, apa yang terjadi di akhirat?" Bagaimanapun juga, ChatGPT pasti punya jawabannya.

BAB 3
ANALISIS PEMECAHAN MASALAH

3.1 Langkah Pemecahan Masalah

Pada penegerjaan tugas kali ini, pemecahan masalah implementasi *chatbot* harus dilakukan dengan terlebih dahulu berfokus untuk mengerjakan *backend*. Pertama-tama, bagian basis data. Data disimpan menggunakan RDBMS MySQL yang memiliki dua buah tabel, yaitu `question` dan `history`. Berikut adalah deskripsi dari tabel-tabel tersebut:

```
mysql> describe question;
+------------+------+------+-----+---------+-------+
| Field      | Type | Null | Key | Default | Extra |
+------------+------+------+-----+---------+-------+
| pertanyaan | text | NO   |     | NULL    |       |
| jawaban    | text | NO   |     | NULL    |       |
+------------+------+------+-----+---------+-------+
mysql> describe history;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| id    | int        | NO   |     | NULL    |       |
| type  | varchar(4) | NO   |     | NULL    |       |
| chat  | text       | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

Gambar 3.1.1 Deskripsi Dua Tabel yang Diimplementasikan
(Sumber: Dokumentasi penulis)

Setiap *query* yang masuk dari pengguna harus dimasukkan ke dalam tabel `history`, demikian pula jawaban yang diberikan oleh *chatbot*. Jika *query*-nya menginstruksikan untuk menambahkan pertanyaan dengan jawaban tertentu, atau menghapus pertanyaan tertentu, barulah kita mengolah isi dari tabel `question`. Berikut adalah contoh isi dari kedua tabel setelah dilakukan beberapa baris percakapan:

```
mysql> select * from history;
+----+------+-----------------------------------------------------------------------------------+
| id | type | chat                                                                              |
+----+------+-----------------------------------------------------------------------------------+
|  0 | bot  | INISIALISASI                                                                      |
|  1 | bot  | Hai, ada yang bisa saya bantu? :)                                                 |
|  1 | me   | hari apa pada tanggal 06-02-2003?                                                 |
|  1 | bot  | Hari Kamis                                                                        |
|  1 | me   | Berapa hasil dari (5^2)+7*6                                                       |
|  1 | bot  | Hasilnya adalah 67                                                                |
|  1 | me   | tambahkan pertanyaan anjing makannya apa? dengan jawaban anjing makannya whiskas  |
|  1 | bot  | Pertanyaan anjing makannya apa? telah ditambahkan...                              |
|  1 | me   | hapus pertanyaan siapa presiden Indonesia?                                        |
|  1 | bot  | Pertanyaan tidak ditemukan!                                                       |
|  1 | me   | apa matkul wajib terseru di semester 4                                            |
|  1 | bot  | yang paling seru adalah STIMA tentunya :D                                         |
|  1 | me   | tambahkan pertanyaan apa tubes terseru stima? dengan jawaban tubes 3 dong...      |
|  1 | bot  | Pertanyaan apa tubes terseru stima? telah ditambahkan...                          |
+----+------+-----------------------------------------------------------------------------------+
```

Gambar 3.1.2 Contoh Isi Tabel `history`
(Sumber: Dokumentasi penulis)

```
mysql> select * from question;
+---------------------------------------+-----------------------------------------+
| pertanyaan                            | jawaban                                 |
+---------------------------------------+-----------------------------------------+
| kucing apa yang ga pernah salah       | kucing ga wrong                         |
| Apa ibu kota Indonesia                | Ibu kota Indonesia adalah Jakarta       |
| apa matkul wajib terseru di semester 4 | yang paling seru adalah STIMA tentunya :D |
| anjing makannya apa?                  | anjing makannya whiskas                 |
| apa tubes terseru stima?              | tubes 3 dong...                         |
+---------------------------------------+-----------------------------------------+
```

Gambar 3.1.2 Contoh Isi Tabel `question`

(Sumber: Dokumentasi penulis)

Atribut `id` pada tabel `history` dimaksudkan untuk menandai nomor *chat*, agar bisa dikelompokkan pada saat menampilkan ke pengguna. Nantinya, akan ditambahkan sebuah fungsi yang bertugas membuat *chat* dengan ID baru. Sedangkan atribut `type`, berisi ”me” yang berarti teks pada kolom `chat` berasal dari pengguna, atau ”bot” yang berarti teks pada kolom `chat` berasal dari *chatbot*.

Diketahui bahwa masukan dari pengguna berupa kata-kata dalam bahasa sehari-hari. Oleh karena itu, masukan tersebut harus diproses terlebih dahulu dengan menggunakan *regular expression* yang dibangun secara manual. Dengan demikian, program Node.js akan tahu pertanyaan jenis apa yang diajukan oleh pengguna, sehingga bisa memberikan jawaban tertentu untuk beberapa fitur spesifik yang diimplementasikan.

Jika masukan pengguna tidak memenuhi *regular expression* mana pun yang didefinisikan, masukan tersebut akan dianggap sebagai pertanyaan sehingga program akan mengambil seluruh data dari tabel `question` pada kolom `pertanyaan` untuk dicocokkan dengan masukan pengguna, menggunakan algoritme KMP/BM (tergantung pilihan pengguna). Apabila ternyata pada tabel terdapat pertanyaan yang *exact match* atau minimal 90% mirip (dihitung menggunakan *similarity percentage*) dengan masukan, maka *chatbot* akan mengembalikan jawaban dari pertanyaan tersebut. Namun jika tidak ada, *chatbot* akan merekomendasikan 3 pertanyaan dari keseluruhan tabel, yang memiliki tingkat kemiripan paling tinggi saat dibandingkan dengan masukan.

Beralih ke sisi transfer data, pengguna akan memasukkan *query*-nya dari *frontend*, kemudian diteruskan ke *backend* untuk diproses sesuai dengan langkah yang baru saja dijelaskan. Setelah *backend* selesai memproses basis data, *frontend* akan mengambil kembali data yang sudah diperbaharui, untuk kemudian ditampilkan ke pengguna.

3.2 Fitur Fungsional dan Arsitektur Aplikasi

Ada beberapa fitur fungsional kritis yang dapat secara langsung dinikmati oleh pengguna selama menggunakan program, yaitu seperti yang sudah disampaikan pada **BAB 1**:

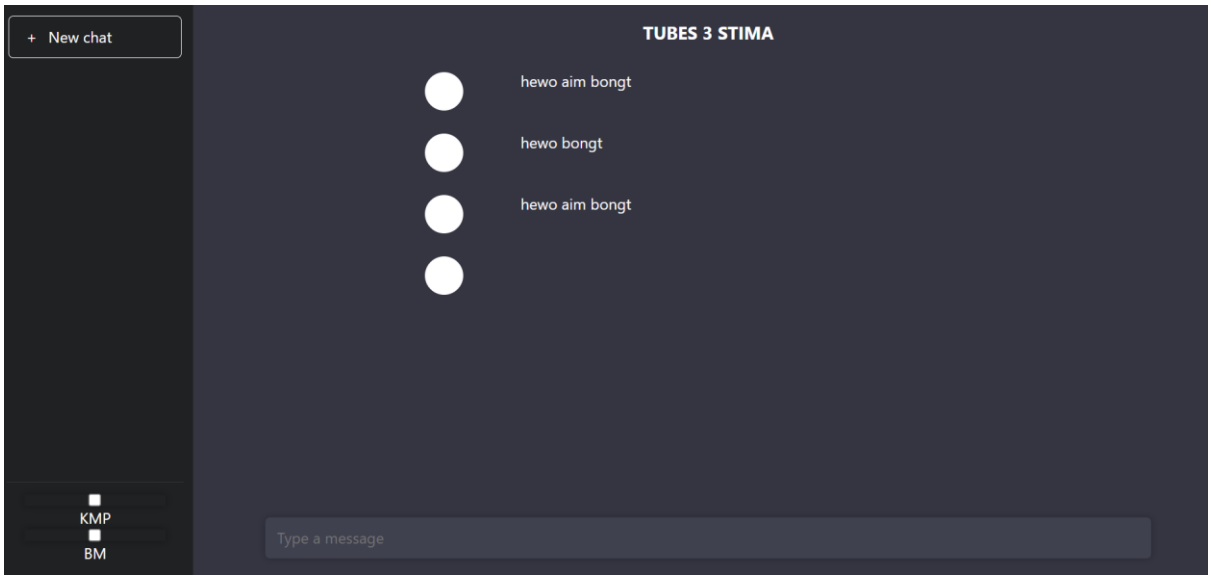
1. Aplikasi dapat menjawab pertanyaan teks yang diajukan pengguna (jawaban didapat dari akses terhadap *database*).
2. Aplikasi dapat menjawab *query* dari pengguna yang berisi persamaan matematika sederhana, yaitu operasi tambah, kurang, kali, bagi, kurung, dan pangkat.
3. Aplikasi dapat memberikan jawaban berupa nama hari dari tanggal yang dimasukkan oleh pengguna.

4. Aplikasi dapat menambahkan suatu pertanyaan ke basis data, dipasangkan dengan jawaban yang dikehendaki oleh pengguna.
5. Aplikasi dapat menghapus pertanyaan tertentu beserta jawabannya dari *database*.

Fitur-fitur lainnya yang mendukung adalah, aplikasi dapat menerima input pengguna, pilihan algoritme KMP atau BM untuk mencocokkan *string* ketika menjawab pertanyaan, serta ”*new chat*” dan *history* sehingga *chat* yang sudah berlalu dapat dilihat kembali. Input pengguna yang dimaksud berupa *string* yang akan diolah oleh *backend* dan dijawab oleh aplikasi *chatbot*.

Fitur yang memanfaatkan akses tabel `question`, algoritme KMP/BM, serta algoritme *similarity percentage* hanyalah fitur 1, 4, dan 5 (mengacu pada nomor di atas). Seperti yang sudah disebutkan sebelumnya, pada fitur 1, akses hanya dilakukan untuk mengambil data, yang kemudian dicocokkan dengan masukan pengguna menggunakan kedua algoritme tersebut. Sedangkan pada fitur 4 dan 5, akses ke tabel akan mengubah data (*insert* / *update* / *delete*) di dalamnya, berdasarkan validasi melalui algoritme-algoritme yang sama. Maksudnya, sebuah pertanyaan akan ditambahkan ke *database* jika belum ada, tetapi jawabannya yang akan di-*update* jika pertanyaan tersebut sudah ada. Pun, sebuah pertanyaan akan dihapus hanya jika ada di dalam *database*. Untuk lebih jelasnya, tersedia pada hasil pengujian di **BAB 4**.

Fitur lainnya seperti kalkulator, tidak perlu melakukan akses ke basis data karena hanya perlu melakukan perhitungan terhadap persamaan matematika yang diberikan oleh pengguna. Demikian pula fitur tanggal. Namun, fitur mana pun yang dijalankan tetap harus menambahkan data ke tabel `history` sebagai langkah pencatatan *chat*.



Gambar 3.2.1 Tampilan Awal dari *Chatbot* yang Dibuat
(Sumber: Dokumentasi penulis)

```
const regexPattern01 = /(.*)?hari (apa )*(pada |di |untuk )*(tanggal )*(\d{1,2})[/\-.](\d{1,2})[/\-.](\d{4})(.*)?/gi;
const regexPattern02 = /(pada )*(tanggal )*(\d{1,2})[/\-.](\d{1,2})[/\-.](\d{4})(.*)?hari (apa)*(.*)?/gi;
const regexPattern03 = /(\d{1,2})[/\-.](\d{1,2})[/\-.](\d{4})/gi;
const regexPattern11 = /(([0-9()])+[\+|\/|\-|\*|\^]+)+([0-9()])*/g;
const regexPattern12 = /([Bb]erapa ([Hh]asil )?)?(([Hh]asil) (dari))? (([0-9()])+[\+|\/|\-|\*|\^]+)+([0-9()])*(\?)?/g;
const regexPattern21 = /Tambahkan pertanyaan .+ dengan jawaban .+/gi;
const regexPattern31 = /Hapus pertanyaan .+/gi;
```

Gambar 3.2.2 *Regular Expression* yang Digunakan untuk Klasifikasi
(Sumber: Dokumentasi penulis)

BAB 4
IMPLEMENTASI DAN PENGUJIAN

4.1 Fungsi-Fungsi Pendukung

No	Nama Fungsi	Penjelasan
1	`BoyerMooreMatch`	Algoritme pencocokan *pattern* pada sebuah *string*, selalu dimulai dari indeks terakhir *pattern*. Jika terdapat ketidakcocokan, ada tiga kasus pergeseran yang perlu diperiksa secara terurut dan ditangani secara berbeda. Teori lebih lanjut dapat dilihat pada bagian **2.4** (halaman 10).
2	`KMP`	Algoritme pencocokan *pattern* pada sebuah *string* yang mana memiliki alur yang mirip seperti algoritme *brute force* namun apabila ada ketidakcocokan, algoritme akan loncat sebanyak *n* huruf. *n* huruf ini diambil dari *border function*.
3	`BorderFunction`	Digunakan untuk menghitung *border funtion* pada sebuah kata yang ingin digunakan pada pencocokan *string*.
4	`levenshteinDistance`	Algoritme ini digunakan untuk menghitung kemiripan pada sebuah *string*.
5	`similarityPercentage`	Lanjutan dari `levenshteinDistance`, digunakan untuk menghitung persentase kemiripan dua buah *string* sesuai dengan hasil `levenshtainDistance`-nya.
6	`getDayOfWeek`	Fungsi ini digunakan untuk me-*return* nama hari dari tanggal yang dimasukkan.
7	`evaluateExpression`	Menghitung operasi matematika dari *string* yang masukan.
8	`answerQuestionBM`	Menjawab pertanyaan masukan dengan menggunakan algoritme BM sebagai algoritme pencocokan *string* masukan dengan pertanyaan pada basis data.
9	`answerQuestionKMP`	Menjawab pertanyaan masukan dengan menggunakan algoritme KMP sebagai algoritme pencocokan *string* masukan dengan pertanyaan pada basis data.
10	`addQuestion`	Menambahkan pertanyaan dan jawaban ke dalam *database*. Apabila pertanyaan sudah ada, fungsi akan meng-*update* jawaban
11	`deleteQuestion`	Menghapus pertanyaan dalam *database* (jika ada yang sesuai dengan masukan)

12	`matchRegEx`	Mengembalikan angka kategori *query* dari *string* yang dimasukkan pengguna. Kategori yang dimaksud adalah: 0 (fitur tanggal), 1 (fitur kalkulator), 2 (fitur tambah pertanyaan), 3 (fitur hapus pertanyaan), -1 (fitur jawab pertanyaan).
13	`proccesQuery`	Mengolah masukan sesuai dengan kategori *string* masukan. Melakukan *parsing* jika diperlukan, serta memanggil fungsi-fungsi seperti `getDayOfWeek`, `addQuestion`, `evaluateExpression`,`answerQuestion`
14	`realProcess`	Gabungan antara `processQuery` dengan kumpulan *regex* yang sudah didefinisikan.

4.2 Cara Menggunakan Program

Program ini menggunakan *backend* dengan bahasa Node.js dan *frontend* dengan bahasa React.js, sehingga untuk menggunakan program dibutuhkan instalasi terhadap beberapa hal berikut :

a. *Code writer* seperti VSCode, yang mana di dalamnya dilakukan instalasi *extentsion* untuk melakukan *run* dan *debug* terhadap bahasa Node.js dan React.js.

b. Program ini memerlukan MySQL sebagai *database* untuk mengolah data.

Setelah melakukan *set up* dan instalasi terhadap kebutuhan program, maka selanjutnya dapat melakukan langkah-langkah berikut ini:

1. Melakukan *clone* terhadap *repository* program ini, pranala dapat dilihat pada bagian **LAMPIRAN**.
2. *Create database* baru pada MySQL, kemudian *load* fail "pertanyaan.sql" yang ada pada folder "data" di *repository*, ke *database* baru tersebut.
3. Ubah bagian "..." pada fail "DatabaseAccess.js" sehingga sesuai dengan MySQL lokal yang sebelumnya ditambahkan *database* baru.

```
const pool = mysql.createPool ({
    host: 'localhost',
    user: '...',
    password: '...',
    database: '...'
}).promise();
```

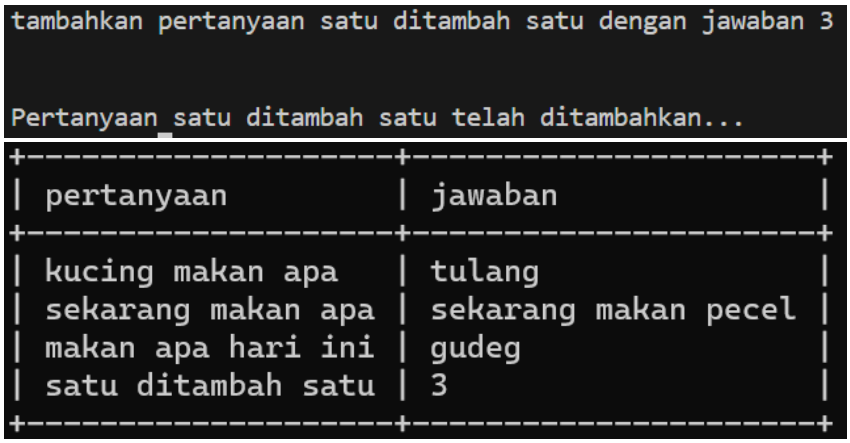
Gambar 4.2.1 Bagian DatabaseAccess.js yang Perlu Perubahan
(Sumber: Dokumentasi penulis)

4. Ganti direktori ke tubes3 kemudian eksekusi perintah berikut pada terminal:
   - `npm run start:frontend`
   - `npm run start:backend`

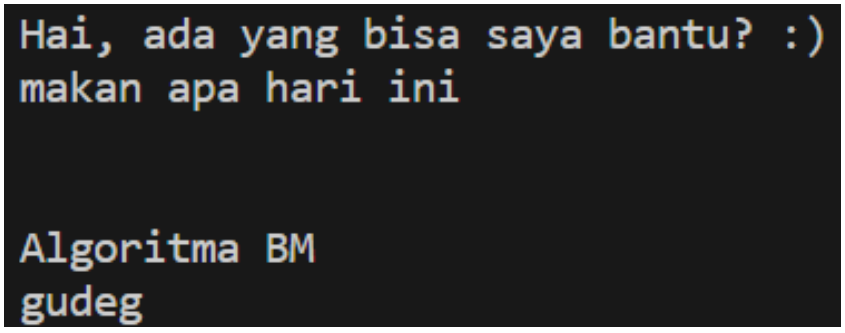
5. Pada *frontend*, pengguna dapat memberikan input yang terkait dengan persamaan matematika dan *date*. Hal ini dikarenakan *frontend* dari program yang kami buat belum dapat terkoneksi dengan baik terhadap *database* sehingga tidak dapat memberikan jawaban terhadap pertanyaan yang jawabannya harus diambil dari *database*.
6. Untuk memberikan pertanyaan yang diharuskan berinteraksi dengan *database*, maka dapat dilakukan secara *command line* pada terminal.

4.3 Pengujian

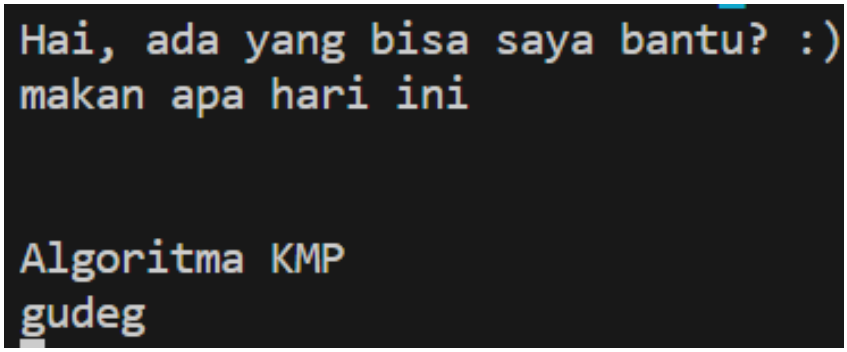
4.3.1 Pengujian dengan *Command-Line*



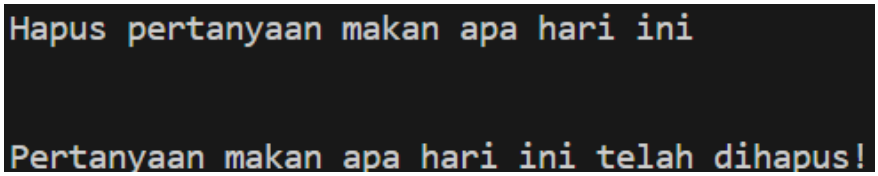
Gambar 4.3.1.1 Tambah Pertanyaan
(Sumber: Dokumentasi penulis)



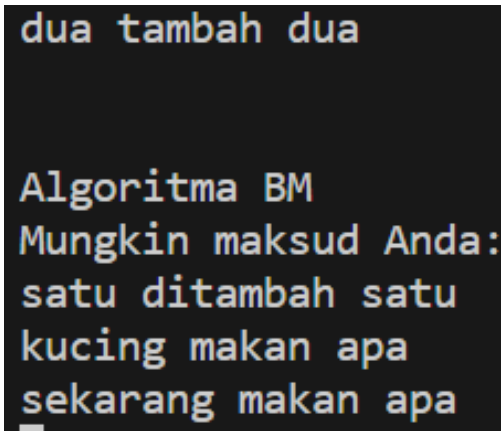
Gambar 4.3.1.2 Tanya Pertanyaan *Exact Match* (BM)
(Sumber: Dokumentasi penulis)



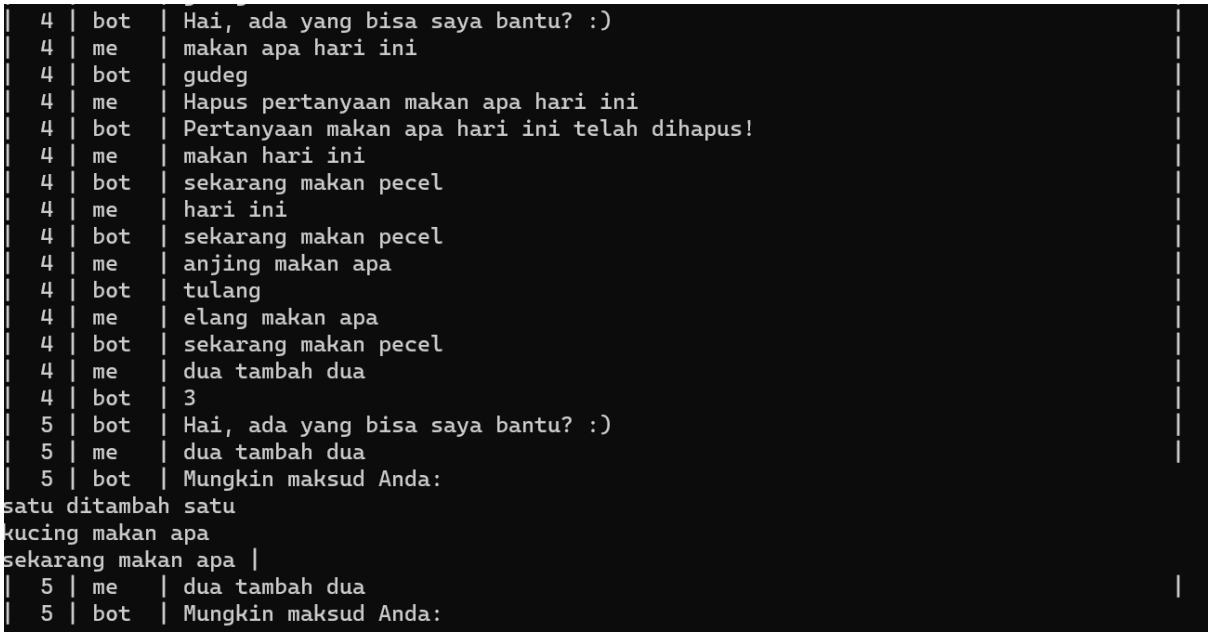
Gambar 4.3.1.3 Tanya Pertanyaan *Exact Match* (KMP)
(Sumber: Dokumentasi penulis)



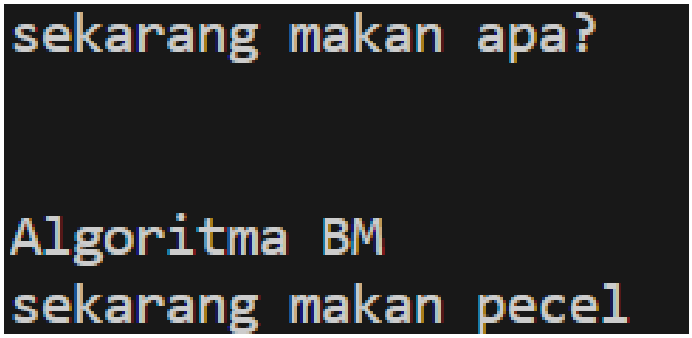
Gambar 4.3.1.3 Hapus Pertanyaan
(Sumber: Dokumentasi penulis)



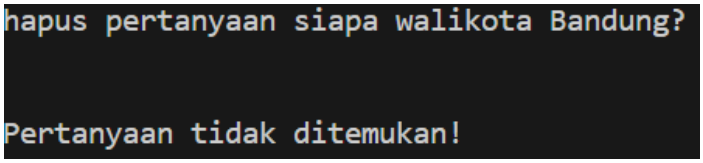
Gambar 4.3.1.4 Jawab Pertanyaan yang Tidak *Match* (<90%)
(Sumber: Dokumentasi penulis)



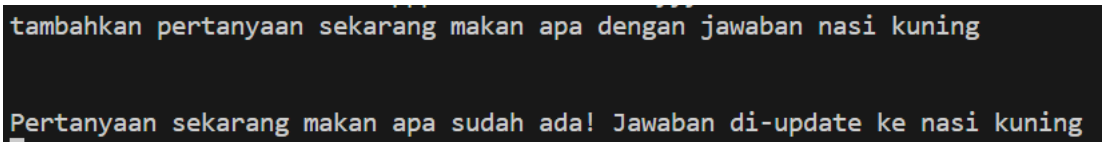
Gambar 4.3.1.5 Kondisi Tabel `history` Setelah Percakapan Panjang
(Sumber: Dokumentasi penulis)



Gambar 4.3.1.6 Jawab Pertanyaan yang Tidak *Exact Match* namun Kemiripan >90%
(Sumber: Dokumentasi penulis)

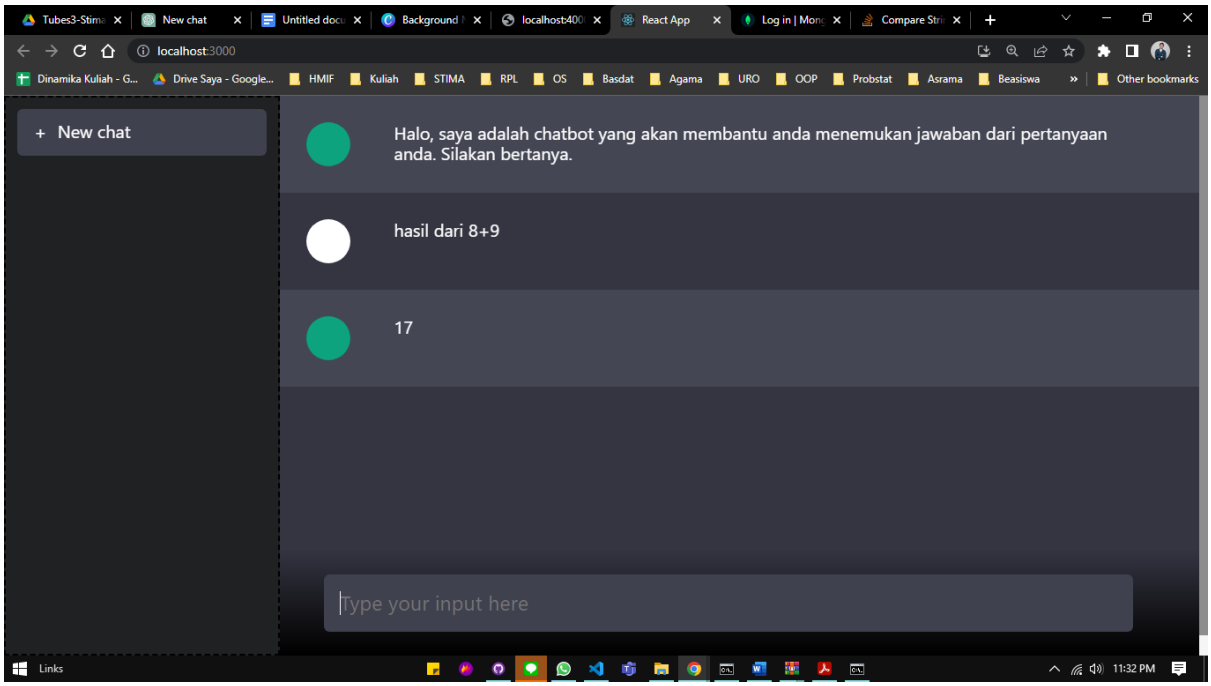


Gambar 4.3.1.7 Hapus Pertanyaan yang Tidak ada di Basis Data
(Sumber: Dokumentasi penulis)

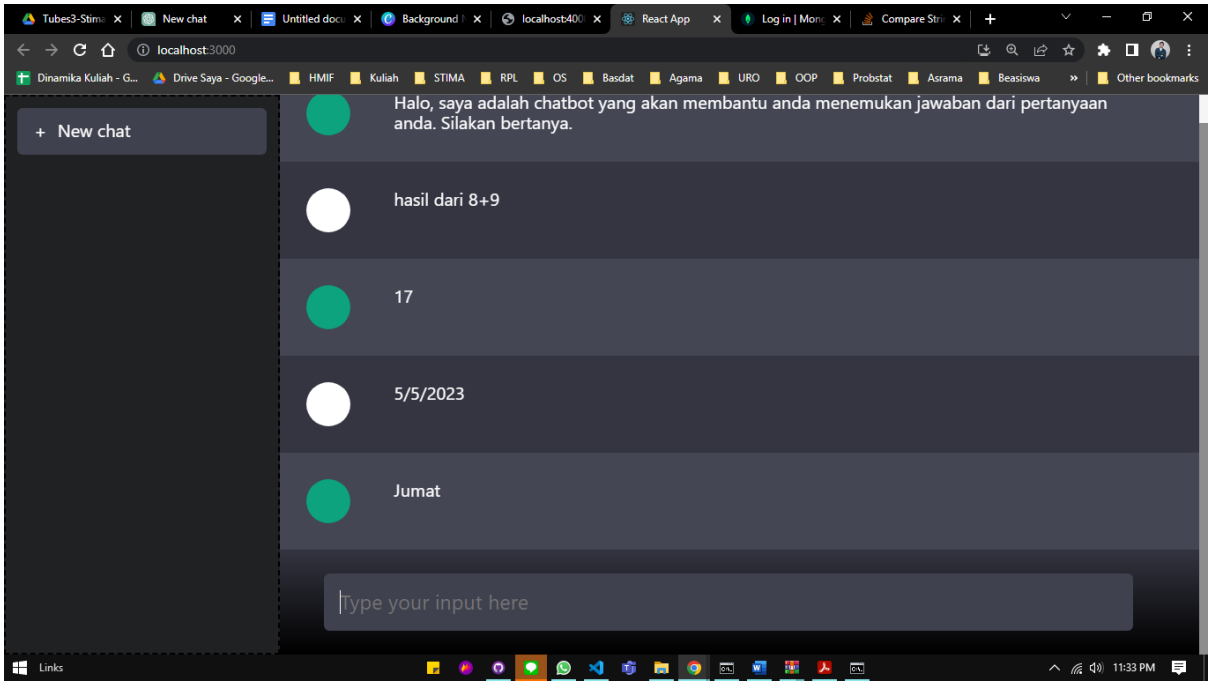


Gambar 4.3.1.8 Tambah Pertanyaan yang Sudah Ada
(Sumber: Dokumentasi penulis)

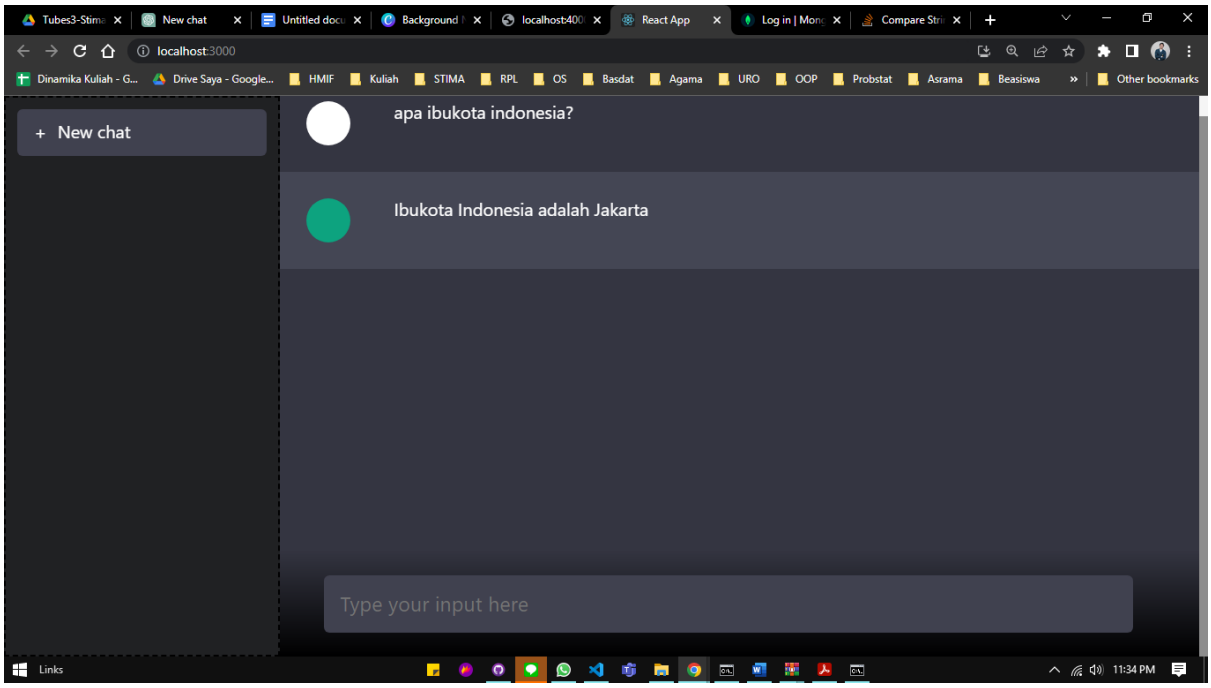
4.3.2 Pengujian Pada *Frontend*



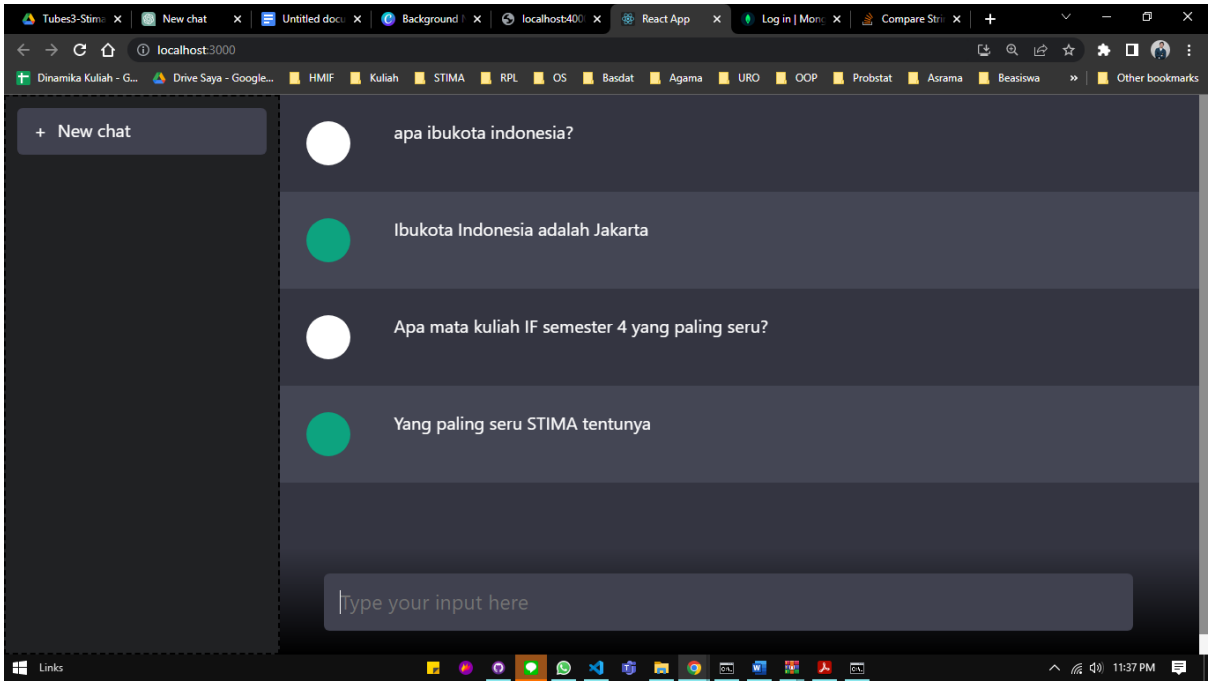
Gambar 4.3.1.9 Fitur Kalkulator pada *Frontend*
(Sumber: Dokumentasi penulis)



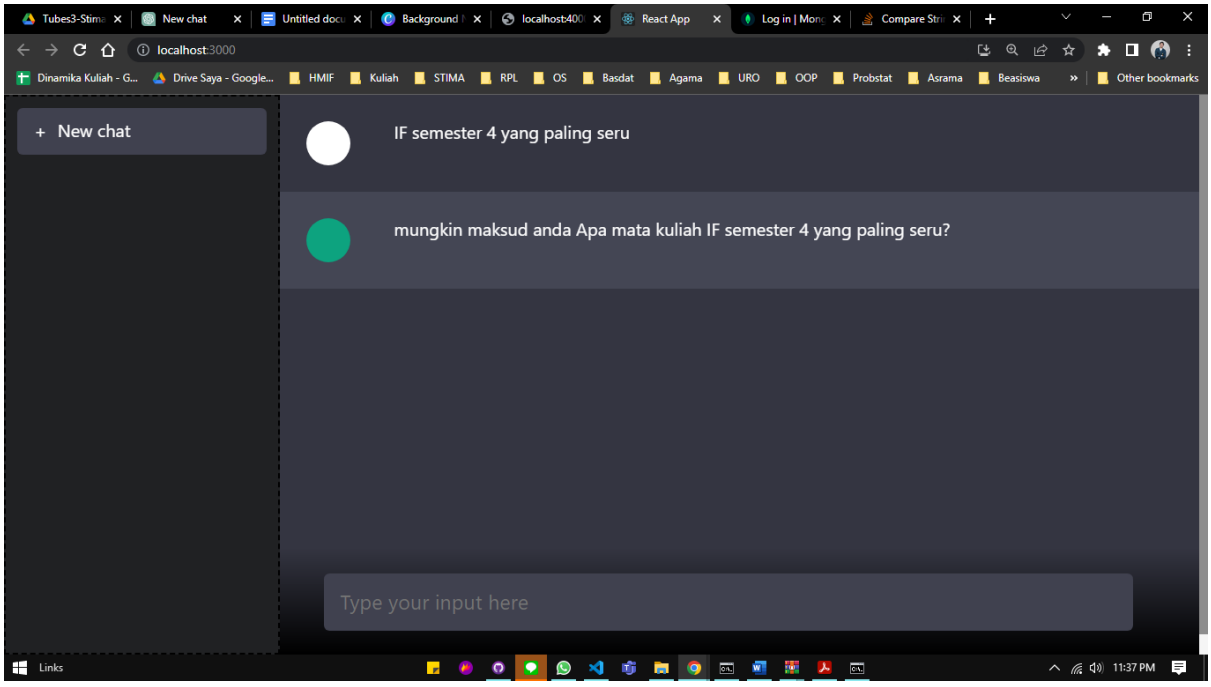
Gambar 4.3.1.10 Fitur Tanggal pada *Frontend*
(Sumber: Dokumentasi penulis)



Gambar 4.3.1.11 Fitur Pertanyaan 1 pada *Frontend*
(Sumber: Dokumentasi penulis)



Gambar 4.3.1.12 Fitur Pertanyaan 2 pada *Frontend*
(Sumber: Dokumentasi penulis)



Gambar 4.3.1.13 Fitur Pertanyaan 3 pada *Frontend*
(Sumber: Dokumentasi penulis)

4.4 Analisis Pengujian

Dibandingkan dengan *brute force*, algoritme KMP dan BM sama-sama dapat memberikan hasil yang lebih optimal. Keduanya juga sudah terbukti mengembalikan hasil yang akurat, yaitu indeks pertama kali *pattern* ditemukan pada sebuah teks.

Meskipun masih belum dapat terintegrasi dengan basis data, *frontend* sudah dapat memberikan jawaban yang tepat untuk *query* meminta nama hari dari tanggal, dan *query* kalkulator. Selebihnya, *query* untuk menambah / menghapus pertanyaan, serta menjawab pertanyaan, baru bisa dilakukan oleh *backend* pada *command-line* di terminal.

Kemiripan minimal 90% jika tidak *exact match*, baru pertanyaan bisa terjawab, mungkin terasa lumayan tinggi, karena pada beberapa kasus, ada perbedaan sedikit saja tidak dapat memberikan jawaban. Akan tetapi, setelah beberapa kali melakukan uji coba, persentase ini merupakan hal yang baik karena perbedaan pertanyaan sedikit saja dapat memberikan jawaban yang tidak seharusnya.

Pada gambar 4.3.1.11-13, bisa dilihat bahwa *frontend* pun akhirnya berhasil dalam memberikan jawaban atas pertanyaan yang butuh akses ke *database*. Dalam kasus ini, *database* yang digunakan adalah MongoDB. Akan tetapi, integrasi kepada algoritme KMP, BM, dan persentase kemiripan, masih belum bisa dilakukan karena keterbatasan waktu.

BAB 5
PENUTUP

5.1 Kesimpulan

Dunia modern tidak lepas dari kehadiran inteligensi buatan (AI / *artificial intelligence*). ChatGPT adalah contoh alat pemrosesan bahasa alami (*natural language*) yang digerakkan oleh teknologi AI, yang memungkinkan penggunanya melakukan percakapan dalam bentuk teks, layaknya percakapan sehari-hari, namun yang menjadi lawan bicaranya adalah sebuah *chatbot*. ChatGPT dapat menjawab pertanyaan dan membantu orang-orang menyelesaikan tugas, seperti menulis email, esai, bahkan kode program.

Di antara sekian banyak algoritme yang ada di dunia, algoritme *string matching*, seperti Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM), ditambah dengan *regular expression*, cocok untuk mendekati persoalan implementasi aplikasi semacam ChatGPT. Dengan berbagai fungsi pendukung yang diimplementasikan, termasuk di dalamnya *similarity percentage* yang melibatkan *Levenshtein distance*, *backend* dan *frontend* dari aplikasi "ChatGPT sederhana" masing-masing 100% berhasil dibuat dengan menggunakan Node.js, MySQL, dan React.js

Meskipun demikian, ada beberapa kendala yang dialami, seperti kesulitan integrasi antara *backend*, *frontend*, dengan *database*. Tim penulis terpaksa menggunakan CLI (*Command-Line Interface*) sebagai antarmuka akhir dari program yang dibuat. Meskipun demikian, semua fitur tetap dapat bekerja sebagaimana mestinya. Dapat disimpulkan bahwa persentase sukses keseluruhan mencapai 70%. Program lengkap dapat dilihat pada pranala di bagian **LAMPIRAN**, sedangkan contoh eksekusi ada di **BAB 4**.

5.2 Saran

1. Sebaiknya, ada sesi diskusi mendalam yang dilakukan bersama dengan seluruh anggota kelompok sebelum memulai pengerjaan tugas. Hal ini bertujuan untuk menyamakan persepsi tim, sehingga tidak ada bagian program yang tidak *connect* saat hasil implementasi digabungkan.
2. Akan lebih baik jika para pemrogram melakukan banyak komunikasi berkala seiring membangun bagian sekecil apa pun dalam program. Dengan demikian, diharapkan tidak ada konflik fungsi umum, atau ketidakseragaman struktur program.
3. Seharusnya, waktu yang diberikan untuk pengerjaan tugas ini lebih banyak, dengan pertimbangan bahwa mahasiswa membutuhkan banyak sekali eksplorasi mandiri. Ditambah lagi, waktu yang tersedia bertabrakan dengan tugas besar lainnya.
4. Alangkah baiknya jika semua hal yang berkaitan dengan *frontend*, termasuk kerterhubungannya dengan *backend*, diajarkan terlebih dahulu kepada mahasiswa, mengingat urusan ini sangat teknis dan sulit untuk dipelajari mandiri dalam durasi yang sangat singkat.
5. Rencana penggunaan *tools*, baik untuk *backend* maupun *frontend*, harus dimatangkan sedari awal, agar tidak memicu perubahan besar-besaran yang disebabkan oleh pergantian *tools* di tengah pengerjaan.

5.3 Refleksi

Tugas besar terakhir di semester 4 ini tampaknya tidak berakhir seindah tugas-tugas sebelumnya. Ada banyak suka dan duka yang kelompok kami lewati bersama-sama, meskipun hasil akhirnya tetap kurang memuaskan.

Pelajaran utama yang bisa diambil dari sini adalah pengaturan waktu. Apa pun yang terjadi, kita tidak boleh menunda-nunda pekerjaan, bahkan harus aktif untuk menjadi penggerak bagi anggota yang lain. Selain itu, kami juga belajar bahwa meiningkatkan kekompakan, konsistensi, kegigihan, serta pengetahuan adalah kunci utama terselesaikannya suatu pekerjaan dengan baik.

Ujung yang seperti ini tidak berarti kami menyerah. Kami berjanji untuk terus bangkit dan berusaha lebih lagi di kemudian hari.

LAMPIRAN

Pranala *Repository*:

https://github.com/satrianababan/Tubes3_13521140.git

Pranala Video Youtube:

...

Tabel A Runut Pengerjaan Tugas

POIN	YA	TIDAK
1. Program berhasil dikompilasi tanpa ada kesalahan.	☑	
2. Program dapat menerima masukan dan menuliskan luaran.	☑	
3. Luaran program sudah benar (jawaban benar).	☑	
4. Bonus melakukan *deployment* dikerjakan.		
5. Bonus pembuatan video dikerjakan.		

DAFTAR REFERENSI

1. Munir, Rinaldi. Tanpa tahun. *Pencocokan String (String/Pattern Matching)*. (Dikases pada tanggal 3 Mei 2023 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf)
2. Khodra, Masayu Leylia. Tanpa tahun. *String Matching dengan Regular Expression*. (Dikases dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf pada tanggal 3 Mei 2023)
3. Muhardian, Ahmad. 25 Agustus 2020. *Apa itu Regex? dan Apa Manfaatnya dalam Pemrograman?*. (Diakses dari https://www.petanikode.com/regex/ pada tanggal 5 Mei 2023)
4. Trivusi. 16 Juli 2022. *Apa itu Algoritma Levenshtein Distance? Berikut Definisinya*. (Diakses tanggal 5 Mei 2023 dari https://www.trivusi.web.id/2022/04/apa-itu-algoritma-levenshtein-distance.html)
5. Mulyana, Iyan dkk. 2014. *Penerapan Algoritma Edit Distance Untuk Pengukuran Kemiripan Antar Dokumen Berbahasa Indonesia*. Seminar Nasional Teknologi Informasi, Komunikasi dan Managemen Palembang-Indonesia, 23 Agustus 2014. (Diakses dari http://eprints.binadarma.ac.id/17009/1/2.%20Finis.pdf pada tanggal 5 Mei 2023)
6. Semah, Benjamin. 5 Desember 2022. *What Exactly is Node.js? Explained for Beginners*. (Diakses dari https://www.freecodecamp.org/news/what-is-node-js/ pada tanggal 5 Mei 2023)
7. B., Richard. 31 Januari 2023. *What is MySQL: MySQL Explained For Beginners*. (Diakses dari https://www.hostinger.com/tutorials/what-is-mysql pada tanggal 5 Mei 2023)
8. Ortiz, Sabrina. 18 April 2023. *What is ChatGPT and why does it matter? Here's what you need to know*. (Diakses dari https://www.zdnet.com/article/what-is-chatgpt-and-why-does-it-matter-heres-everything-you-need-to-know/ pada tanggal 5 Mei 2023)