

Tugas Besar 3 IF2211 Strategi Algoritma

Semester II tahun 2022/2023

## **Penerapan String Matching dan Regular Expression dalam Pembuatan ChatGPT Sederhana**

Disusun Oleh:

Kelompok 25 Athena

Noel Christoffel Simbolon 13521096

Michael Jonathan Halim 13521124

Raynard Tanadi 13521143



**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2023**

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I DESKRIPSI TUGAS</b>	<b>4</b>
1.1 Deskripsi Tugas	4
<b>BAB II LANDASAN TEORI</b>	<b>6</b>
2.1. KMP Algorithm	6
2.2. BM Algorithm	8
2.3. Regex	10
2.3.1. Kalkulator	10
2.3.2. Tanggal	10
2.3.3. Tambah Pertanyaan	10
2.3.4. Hapus Pertanyaan	11
2.3.5. Pertanyaan Kosong	11
2.4. Penjelasan Singkat Mengenai Aplikasi Web yang Dibangun	11
<b>BAB III ANALISIS PEMECAHAN MASALAH</b>	<b>13</b>
3.1. Langkah Pemecahan Masalah	13
3.2. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun	14
3.2.1 Front-end	14
3.2.2 Back-end	29
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN</b>	<b>38</b>
4.1. Spesifikasi Teknis Program	38
4.1.1 Struktur Data	38
4.1.2 Fungsi dan Prosedur	40
4.2. Tata Cara Penggunaan Sistem	60
4.3. Hasil Pengujian	61
4.3.1 Pengujian 1	61
4.3.2 Pengujian 2	61
4.3.3 Pengujian 3	62
4.3.4 Pengujian 4	62
4.3.5 Pengujian 5	63
4.3.6 Pengujian 6	63
4.3.7 Pengujian 7	64
4.3.8 Pengujian 8	64
4.3.9 Pengujian 9	64
4.3.10 Pengujian 10	65
4.3.11 Pengujian 11	66
4.4. Analisis Hasil Pengujian	66
<b>BAB V PENUTUP</b>	<b>69</b>
5.1. Kesimpulan	69

5.2. Saran	69
5.3. Refleksi dan Tanggapan	69
<b>DAFTAR PUSTAKA</b>	<b>71</b>
<b>LAMPIRAN</b>	<b>72</b>

# 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). 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.

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 pertanyaan xxx dengan jawaban yyy”. Menggunakan algoritma string matching untuk mencari tahu apakah pertanyaan sudah ada. Apabila sudah, maka jawaban akan diperbarui.

##### 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).

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.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1. KMP Algorithm**

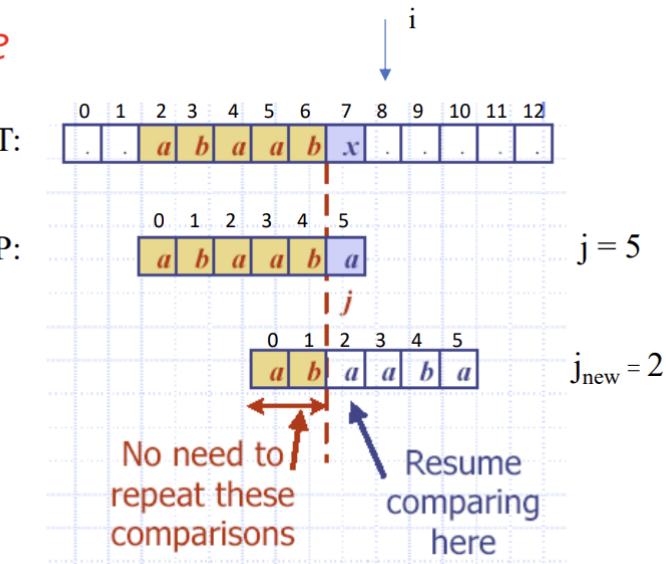
Algoritma KMP atau yang biasa dikenal dengan Algoritma Knuth Morris Pratt merupakan salah satu algoritma yang digunakan untuk melakukan pencocokan string maupun pola. Algoritma ini melakukan pencarian dengan urutan dari kiri ke kanan. Berbeda dengan algoritma *brute force*, algoritma ini dapat mengingat beberapa perbandingan yang dilakukan sebelumnya sehingga dapat melakukan pergeseran yang lebih besar dan efektif yang mengurangi jumlah perbandingan dan meningkatkan kecepatan pencarian.

Pada algoritma KMP, jika terdapat ketidakcocokan antara pola dengan teks yang ada pada indeks - j, maka akan dilakukan pencarian pola terpanjang antara prefix yang sama dengan pola suffix dari pola[0 .. j-1]. Misalkan terdapat pola berupa ‘abaab’, maka berikut adalah langkah untuk mencari pasangan terpanjang prefix dan suffix tersebut :

- Untuk ‘a’, tidak terdapat prefix maupun suffix.
  - Untuk ‘ab’, terdapat prefix berupa ‘a’ dan suffix berupa ‘b’.
  - Untuk ‘aba’, terdapat prefix berupa ‘a’, ‘ab’ dan suffix berupa ‘ba’, ‘a’.
- Terdapat pola yang sama, yaitu ‘a’ dengan panjang 1
- Untuk ‘abaa’, terdapat prefix berupa ‘a’, ‘ab’, ‘aba’ dan suffix berupa ‘baa’, ‘aa’, dan ‘a’.
  - Untuk ‘abaab’, terdapat prefix berupa ‘a’, ‘ab’, ‘aba’, ‘abaa’ dan suffix berupa ‘baab’, ‘aab’, ‘ab’, dan ‘b’. Terdapat pola yang sama, yaitu ‘ab’ dengan panjang 2.

Pola terbesar ini yang merupakan ‘ab’ dengan panjang 2 tidak perlu untuk dicek kembali sehingga pola dapat digeser sebesar 2 dan mulai melakukan pencocokan kembali dari indeks tersebut.

### Example



**Gambar 2.1.1** Ilustrasi Algoritma KMP

Sumber :

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

Algoritma KMP juga memiliki *border function*  $b(k)$  yang didefinisikan sebagai ukuran terbesar dari pola $[0 .. k]$  yang juga merupakan suffix dari pola $[1 .. k]$ .

$$(k = j-1)$$

$j$	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

$b(k)$  is the size of the largest border.

**Gambar 2.1.2** Ilustrasi Border Function

Sumber :

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

Algoritma ini memiliki kompleksitas waktu sebesar  $O(m + n)$  yang terdiri dari menghitung fungsi pinggiran  $O(m)$  dan pencarian string  $O(n)$ . Algoritma ini bagus untuk memproses file yang sangat besar, tetapi tidak bagus untuk ukuran karakter yang dinamis.

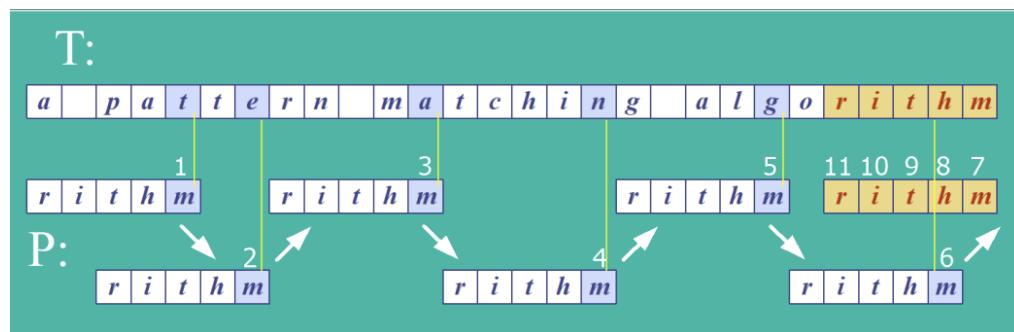
## 2.2. BM Algorithm

Algoritma BM atau yang biasa dikenal dengan Algoritma Boyer Moore merupakan salah satu algoritma yang digunakan untuk pencocokan string maupun pola. Algoritma ini berdasar pada dua teknik, yaitu *looking-glass technique* dan *character-jump technique*.

Pada teknik pertama, pencocokan dilakukan dari karakter pada indeks pola terakhir dengan suatu karakter pada teks. Jika karakter sama, maka pencocokan akan kembali dilakukan dengan berjalan mundur pada pola dan teks.

Pada teknik kedua, jika saat pencocokan terjadi perbedaan karakter antara pola dan teks, maka akan dilakukan satu dari tiga aksi berikut, yaitu mencari karakter pada pola yang indeksnya lebih kecil dari saat terjadi perbedaan dan menggesernya sehingga sejajar, menggeser sebanyak 1 jika karakter tersebut terdapat di dalam pola tetapi indeksnya lebih besar dari saat terjadi perbedaan, atau melakukan pergeseran sejumlah total karakter pada pola jika karakter yang berbeda pada string tidak terdapat pada pola.

### Boyer-Moore Example (1)



Jumlah perbandingan karakter: 11 kali

### Gambar 2.2.1 Ilustrasi Algoritma BM

Sumber :

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

Algoritma ini juga memiliki *last occurrence function* yang berisi informasi mengenai indeks kemunculan terakhir sebuah karakter pada teks di pola. Jika suatu karakter pada teks tidak terdapat pada pola, maka indeksnya akan diberi -1. Fungsi ini akan sangat berguna pada teknik kedua.

### L() Example

- $A = \{a, b, c, d\}$
- $P$ : "abacab"

P	a	b	a	c	a	b
	0	1	2	3	4	5



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

$L()$  stores indexes into  $P[]$

### Gambar 2.2.2 Ilustrasi Last Occurrence Function

Sumber :

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

Algoritma ini bagus untuk digunakan pada string yang memiliki karakter bervariasi, seperti bahasa Inggris. Algoritma ini juga memiliki kompleksitas waktu maksimum sebesar  $O(mn + A)$  di mana  $A$  adalah karakter teks yang ada pada pola.

## 2.3. Regex

### 2.3.1. Kalkulator

```
Regex fitur kalkulator : /^(?:(?:Berapa|Hitung|Kalkulasi)\s+)?([()d+\-*/.^s]+)(\?)?$/i
```

Pola regex ini digunakan untuk menentukan apakah pesan dari pengguna merupakan fitur kalkulator atau tidak. Pola regex dimulai dengan tiga opsional kata kunci yaitu kata “Berapa”, “Hitung”, dan “Kalkulasi”. Kemudian, diikuti oleh ekspresi matematika yang hanya mengandung angka, spasi, operator tambah, kurang, kali, bagi, pangkat, dan tanda kurung. Terakhir, pola diakhiri dengan huruf opsional tanda tanya.

### 2.3.2. Tanggal

```
Regex fitur tanggal : /^(d{1,2})\/(d{1,2})\/(d{4})$/
```

Pola regex ini digunakan untuk menentukan apakah pesan dari pengguna merupakan fitur tanggal atau tidak. Pola regex dimulai dengan angka yang terdiri dari satu atau dua digit untuk tanggal, diikuti dengan garis miring, lalu angka yang terdiri dari satu atau dua digit juga untuk bulan, diikuti dengan garis miring lagi, dan empat digit untuk tahun.

### 2.3.3. Tambah Pertanyaan

```
Regex tambah pertanyaan : /^Tambahkan pertanyaan .+ dengan jawaban .+$/i
```

Pola regex ini digunakan untuk menentukan apakah pesan dari pengguna merupakan fitur menambah pertanyaan ke database atau tidak. Pola regex dimulai dengan kata kunci “Tambahkan pertanyaan” sehingga pesan harus dimulai dengan kata kunci tersebut, diikuti dengan spasi, lalu pertanyaan yang ingin ditambahkan, diikuti dengan spasi, lalu dilanjutkan dengan kata kunci “dengan jawaban”, diikuti dengan spasi, dan terakhir adalah jawaban dari pertanyaan yang ingin ditambahkan.

#### **2.3.4. Hapus Pertanyaan**

```
Regex hapus pertanyaan : /^Hapus pertanyaan .+$/  
i
```

Pola regex ini digunakan untuk menentukan apakah pesan dari pengguna merupakan fitur menghapus pertanyaan dari database. Pola regex dimulai dengan kata kunci “Hapus pertanyaan” yang artinya pesan dari pengguna harus dimulai dengan kata kunci tersebut lalu diikuti dengan pertanyaan yang ingin dihapus oleh pengguna.

#### **2.3.5. Pertanyaan Kosong**

```
Regex pertanyaan kosong : /^\\s*$/  
i
```

Pola regex ini digunakan untuk menentukan apakah sebuah pesan dari pengguna merupakan pesan yang kosong atau tidak. Pesan yang kosong belum tentu karena panjang dari pesan tersebut bernilai nol, namun bisa saja pengguna mengetik spasi, *tab*, atau *newline* saja.

### **2.4. Penjelasan Singkat Mengenai Aplikasi Web yang Dibangun**

Aplikasi web ini dibuat dengan TypeScript, bahasa pemrograman turunan JavaScript dengan fitur tambahan seperti tipe data yang lebih kuat dan dukungan untuk OOP (Pemrograman Berorientasi Objek). Framework Next.js digunakan untuk mengembangkan aplikasi web sisi klien (front-end) yang memungkinkan pengembangan situs web responsif dan lebih mudah dipelihara.

Saat ini, framework Express.js digunakan untuk pengembangan aplikasi web backend. Express.js adalah salah satu framework paling populer untuk mengembangkan aplikasi web dengan Node.js dan memudahkan untuk membangun RESTful API dan melakukan perutean server. Express.js memungkinkan aplikasi web untuk mengakses dan memproses data dari database.

Prisma ORM digunakan sebagai lapisan koneksi antara aplikasi web dan database. Prisma memudahkan developer untuk melakukan operasi CRUD (Create, Read, Update, Delete) pada database dan menyediakan fitur seperti migrasi dan seeding database. Database PostgreSQL dipilih sebagai sistem manajemen database untuk situs web ini. PostgreSQL adalah basis data open source yang dikenal dengan kinerja yang baik, mampu menangani data yang kompleks dan mendukung banyak tipe data dan indeks.

Aplikasi web ini di-deploy pada dua host yang berbeda, yaitu Vercel dan Railway. Vercel adalah platform yang menawarkan layanan hosting gratis untuk aplikasi web front-end. Sementara itu, Railway adalah platform yang menawarkan layanan hosting gratis untuk aplikasi web dan database back-end. Dengan menggunakan dua host yang berbeda, kami dapat memisahkan komponen front-end dan back-end secara terpisah, yang dapat mempermudah deployment dan maintenance.

## **BAB III**

### **ANALISIS PEMECAHAN MASALAH**

#### **3.1. Langkah Pemecahan Masalah**

Untuk memecahkan permasalahan pada tugas besar ini, terdapat beberapa langkah, yaitu

1. Setelah pengguna mengirimkan pesan pada website, pesan dikirimkan ke web server yang akan diolah sesuai algoritma pada back-end.
2. Untuk menentukan jawaban dari bot, pesan akan diklasifikasikan terlebih dahulu. Dengan menggunakan definisi regex yang sudah dibahas pada subbab 2.3, pesan pengguna akan diklasifikasikan apakah pertanyaan tersebut merupakan fitur kalkulator, tanggal, tambah pertanyaan, hapus pertanyaan, pesan kosong, atau bertanya.
3. Setelah pesan pengguna berhasil diklasifikasi, maka pesan akan diolah sesuai dengan klasifikasinya.
4. Jika pesan kosong, maka bot akan menjawab “Pertanyaan kosong.”.
5. Untuk kalkulator, pesan akan diambil cukup persamaan matematikanya saja, lalu diproses menggunakan algoritma kalkulator yang memanfaatkan struktur data stack. Apabila persamaan tidak dapat dihitung, maka bot akan menjawab sintaks persamaan tidak sesuai.
6. Untuk tanggal, pesan akan diambil cukup tanggalnya saja, lalu diproses menggunakan algoritma “Zeller’s Rules” untuk mendapatkan hari dari tanggal tersebut.
7. Untuk tambah pertanyaan, pesan akan diambil pertanyaan yang ingin ditambahkan dan diperiksa terlebih dahulu apakah sudah terdapat pada database apa belum. Jika belum, maka pertanyaan dan jawaban akan disimpan dalam database. Jika sudah, maka jawaban dari pertanyaan tersebut akan di-update dengan jawaban yang baru.
8. Untuk hapus pertanyaan, pesan akan diambil cukup pertanyaannya saja dan diperiksa apakah terdapat pada database atau tidak. Jika iya, maka pertanyaan dan

jawaban akan dihapus dari database. Jika tidak, maka bot akan menjawab pertanyaan tidak terdapat dalam database.

9. Untuk pertanyaan biasa, pesan akan diproses dengan algoritma string matching terlebih dahulu. Akan dicari pertanyaan yang exact match dengan yang ada di database. Apabila ditemukan yang exact match tepat satu, maka akan dijawab oleh bot dengan jawaban yang terdaftar. Apabila ditemukan lebih dari satu, maka akan dihitung persentase kemiripan dengan levenshtein distance dan diambil yang persentase kemiripannya tertinggi.
10. Apabila tidak terdapat pertanyaan yang exact match, maka akan masuk ke algoritma string similarity dengan algoritma levenshtein distance untuk menghitung persentase kemiripan. Jika terdapat persentase kemiripan yang lebih dari 90%, maka diambil yang tertinggi. Jika tidak ada, maka bot akan memberikan maksimal 3 opsi pertanyaan yang memiliki persentase kemiripan di atas 50%.
11. Jika tidak terdapat opsi pertanyaan yang lebih dari 50% (semua persentase kemiripan di bawah 50%), maka bot akan menjawab tidak ada pertanyaan yang terdaftar dalam database.
12. Jika pesan sudah berhasil diproses, maka pesan akan dikirimkan kembali ke aplikasi dari web server.

Untuk prioritas klasifikasi dilakukan dengan memeriksa pertanyaan kosong atau tidak, kalkulator, tanggal, tambah pertanyaan, hapus pertanyaan, dan terakhir menjawab pertanyaan.

### **3.2. Fitur Fungsional dan Arsitektur Aplikasi Web yang Dibangun**

#### **3.2.1 Front-end**

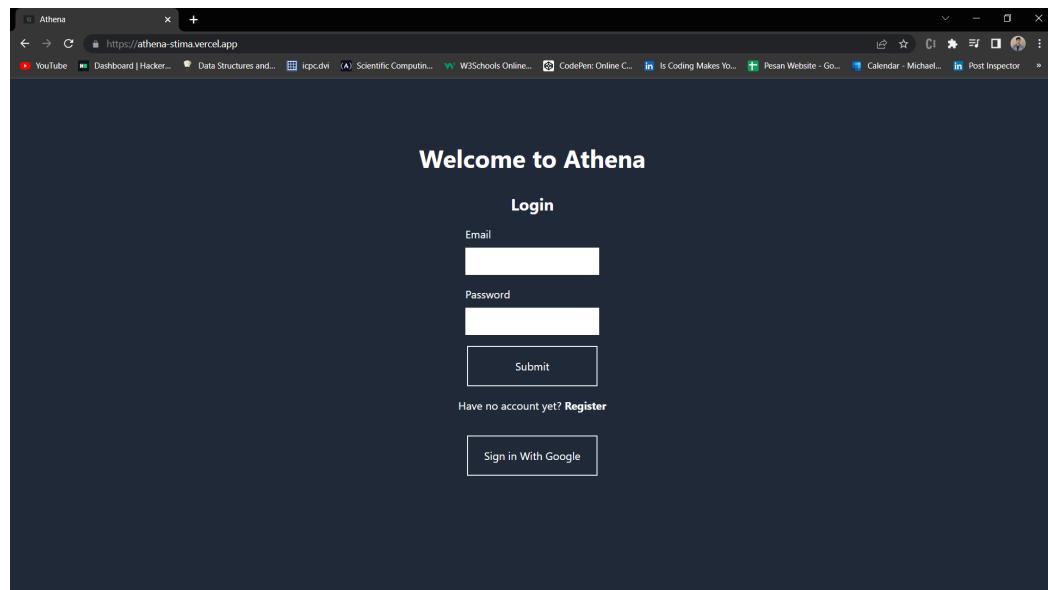
Untuk arsitektur aplikasi web secara front-end, terdapat 4 directory utama dalam project, yaitu public (menyimpan icon dan gambar-gambar yang diperlukan untuk kebutuhan web), styles (menyimpan styling utama untuk aplikasi), pages (menyimpan

file-file halaman yang tersedia pada aplikasi), dan components (menyimpan komponen-komponen yang telah dipecah dari suatu desain abstrak).

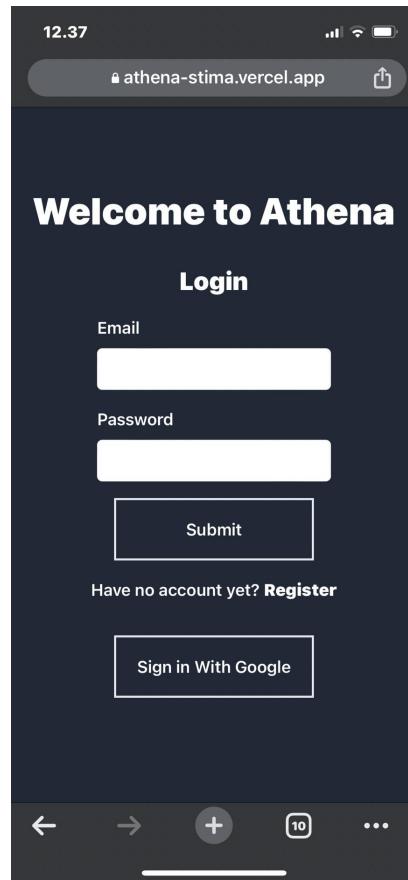
Terdapat beberapa fitur fungsional yang dibangun pada aplikasi dari sisi client (front-end). Fitur-fitur fungsional ini merupakan interaksi pengguna dengan UI aplikasi, seperti

### 1. Halaman Login

Halaman login merupakan halaman yang digunakan untuk melakukan proses autentikasi sebab setiap user akan memiliki sesi/history masing-masing yang bersifat privasi juga sehingga diperlukan autentikasi untuk membuktikan kepemilikan chat. Untuk login sendiri, disediakan dua opsi, yaitu dengan akun biasa atau dengan akun google. Jika menggunakan akun biasa, diperlukan melakukan proses registrasi.



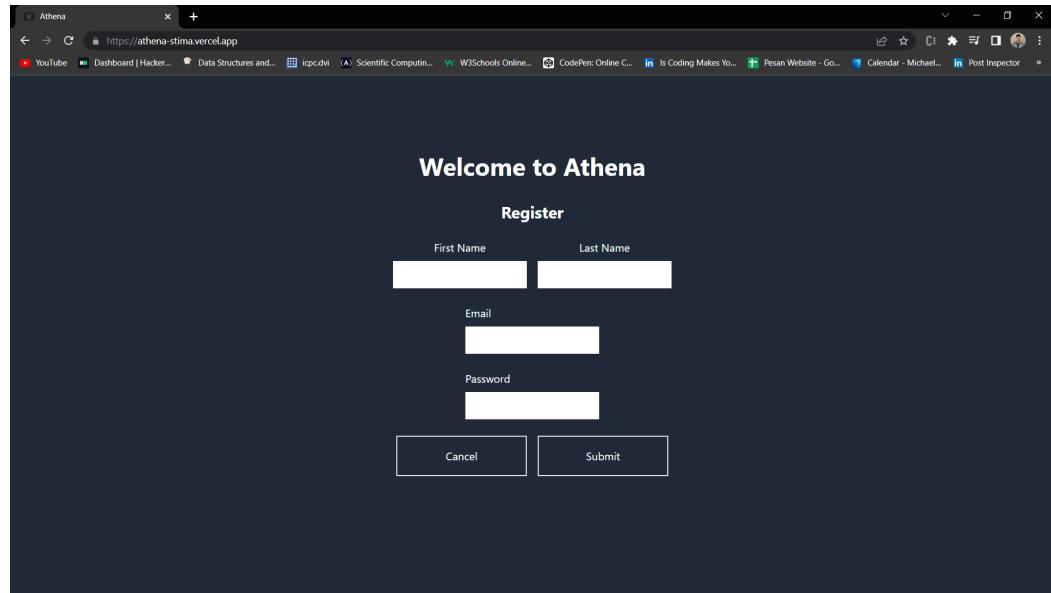
**Gambar 3.2.1.1** Halaman Login dalam Desktop View



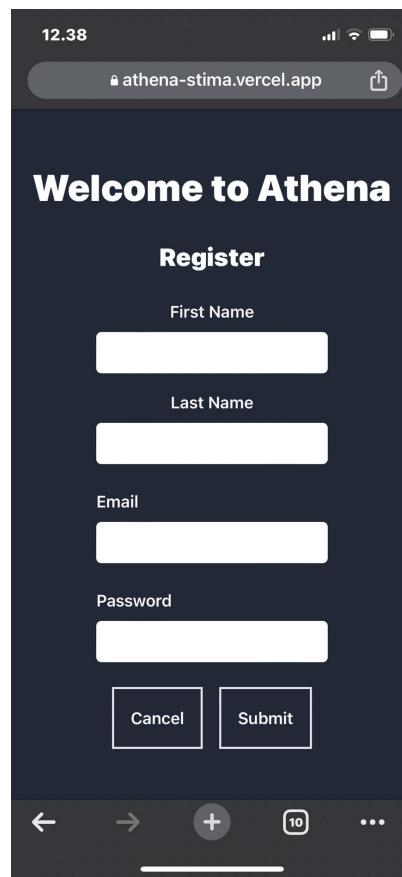
**Gambar 3.2.1.2** Halaman Login dalam Mobile View

## 2. Halaman Register

Halaman register merupakan halaman yang digunakan pengguna untuk membuat akun biasa. Informasi yang harus diisi adalah nama depan, nama belakang, email, dan password.



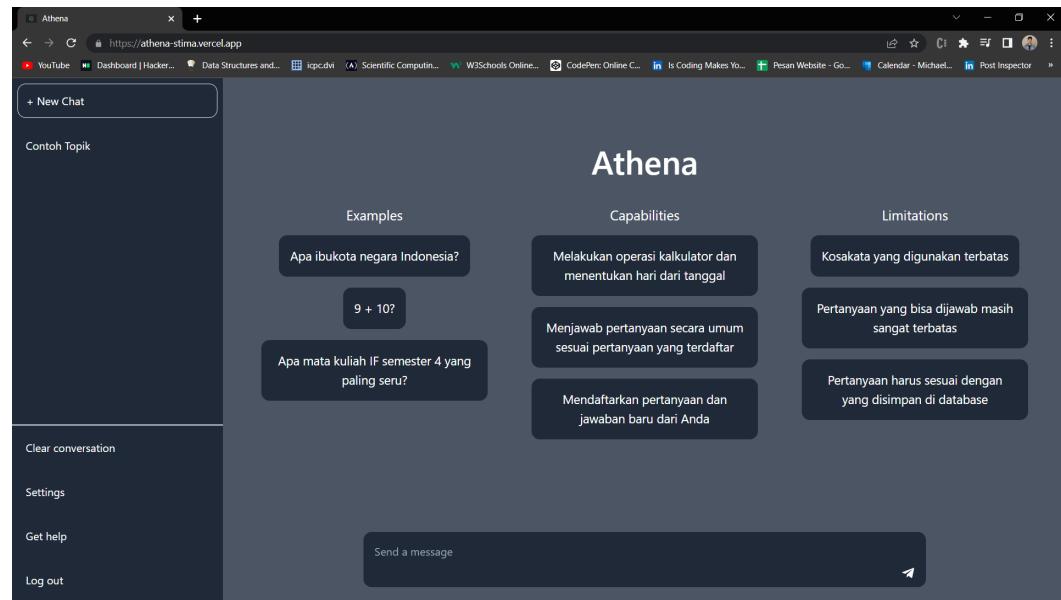
Gambar 3.2.1.3 Halaman Register dalam Desktop View



Gambar 3.2.1.4 Halaman Register dalam Mobile View

### 3. Halaman Utama Aplikasi

Halaman utama disini berisi mengenai penjelasan aplikasi. Halaman ini ditampilkan jika pengguna ingin membuat sesi chat baru atau belum pernah mengirimkan pesan sama sekali.



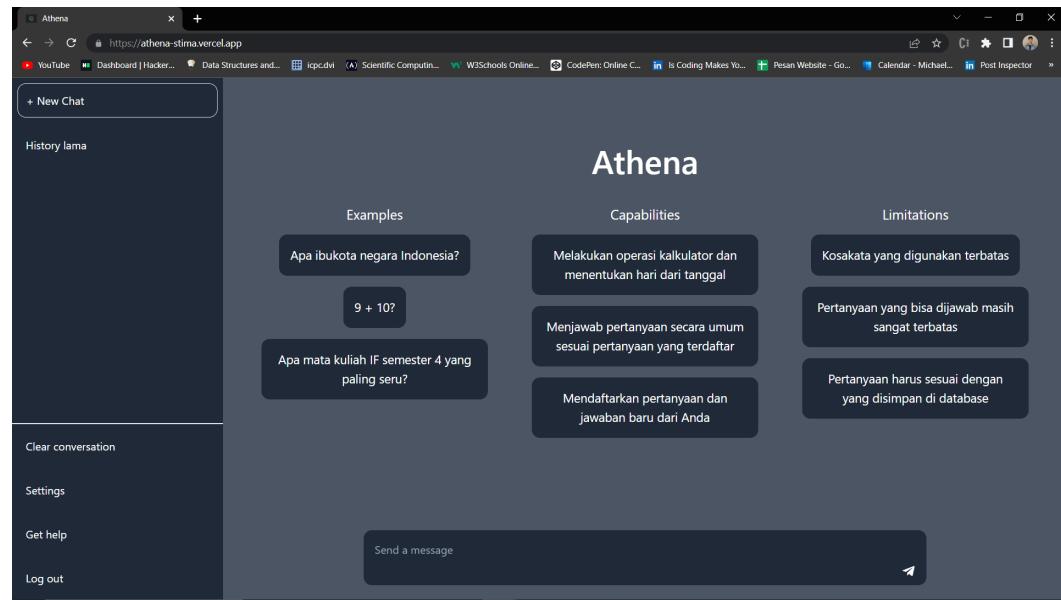
**Gambar 3.2.1.5** Halaman Utama dalam Desktop View



**Gambar 3.2.1.6** Halaman Utama dalam Mobile View

4. Menambah sesi/*history* baru

Pengguna dapat membuat sesi/*history* baru dengan memencet tombol “+ New Chat” pada kiri atas yang dapat dilihat pada gambar di bawah. Untuk tampilan mobile, menu di sebelah kiri akan ditampilkan jika menekan tombol sidebar di kiri atas.



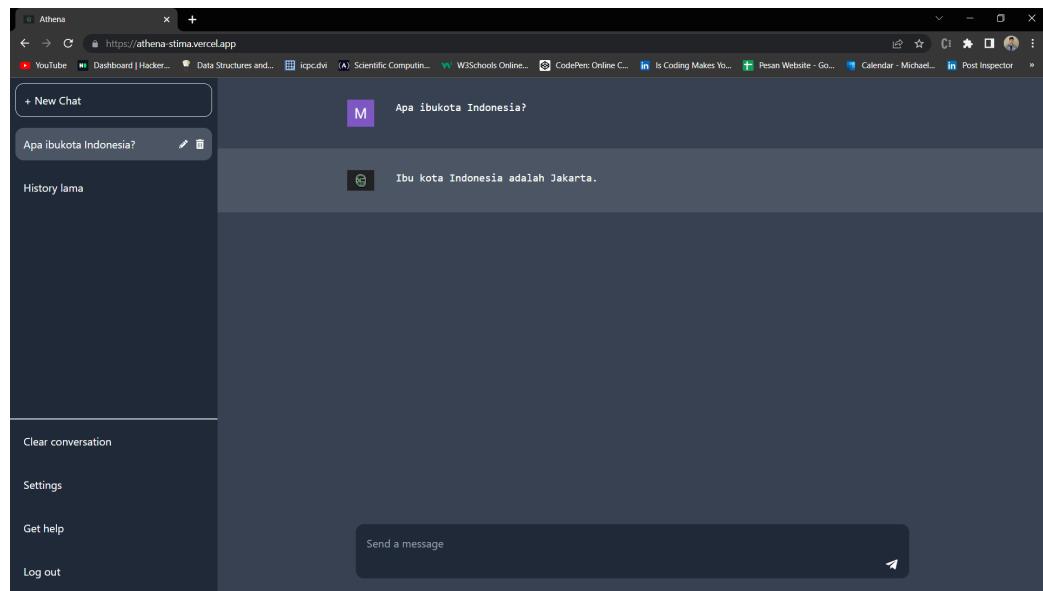
Gambar 3.2.1.7 Menambah Sesi/*History* dalam Desktop View



Gambar 3.2.1.8 Menambah Sesi/*History* dalam Mobile View

##### 5. Mengirim dan menerima pesan dari/ke bot

Pengguna tentu dapat mengirimkan pesan kepada bot dengan mengetikkan pesan di kotak bawah dan menekan enter pada keyboard atau menekan tombol submit. Untuk mengetik newline, perlu menggunakan Shift + Enter pada desktop. Namun, untuk mobile view berbeda behavior pengetikan keyboardnya karena tidak ada shift sehingga untuk mengirim pesan harus menekan tombol submit.



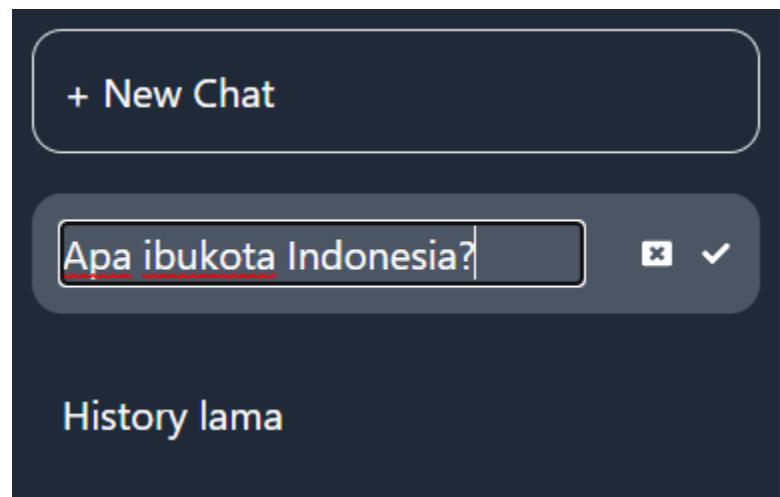
**Gambar 3.2.1.9** Mengirim Pesan dalam Desktop View



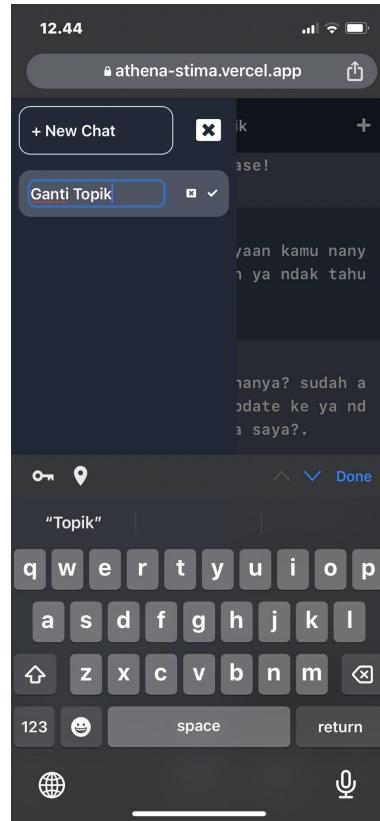
**Gambar 3.2.1.10** Mengirim Pesan dalam Mobile View

6. Mengganti topik/judul dari setiap sesi/history

Pengguna dapat mengganti isi dari topik/judul untuk setiap sesi/history dengan menekan tombol pensil pada kotak history dan mengubah isi dari topik/judul tersebut dan menekan tombol centang untuk konfirmasi perubahan.



**Gambar 3.2.1.11** Mengganti Topik Sesi/*History* dalam Desktop View

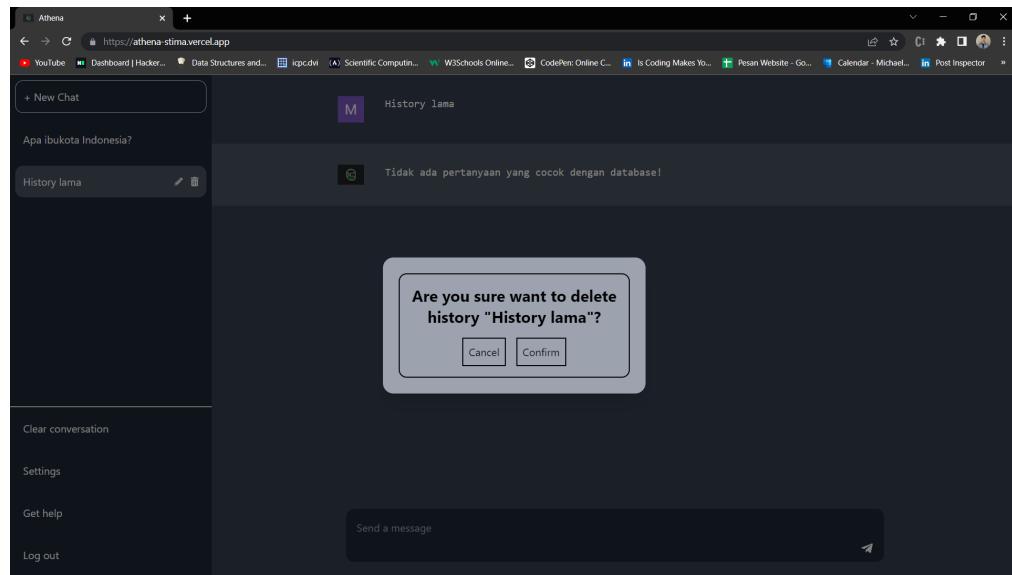


**Gambar 3.2.1.12** Mengganti Topik Sesi/*History* dalam Mobile View

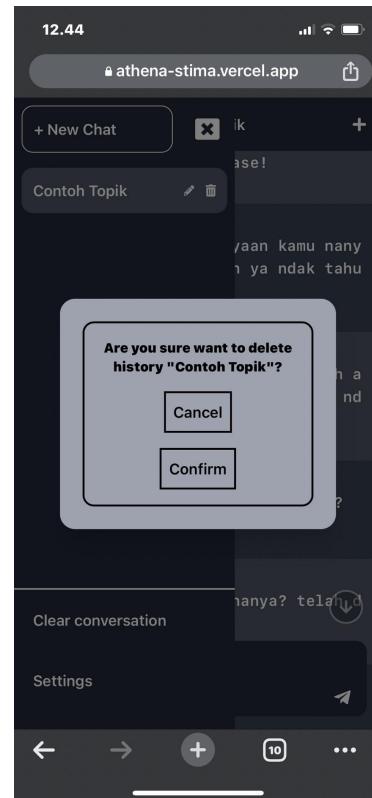
7. Menghapus masing-masing sesi/*history*

Pengguna dapat menghapus masing-masing sesi/history dengan menekan tombol hapus pada kotak history dan akan muncul popup konfirmasi penghapusan. Jika

yakin ingin dihapus, maka pengguna akan menekan tombol “Confirm”.



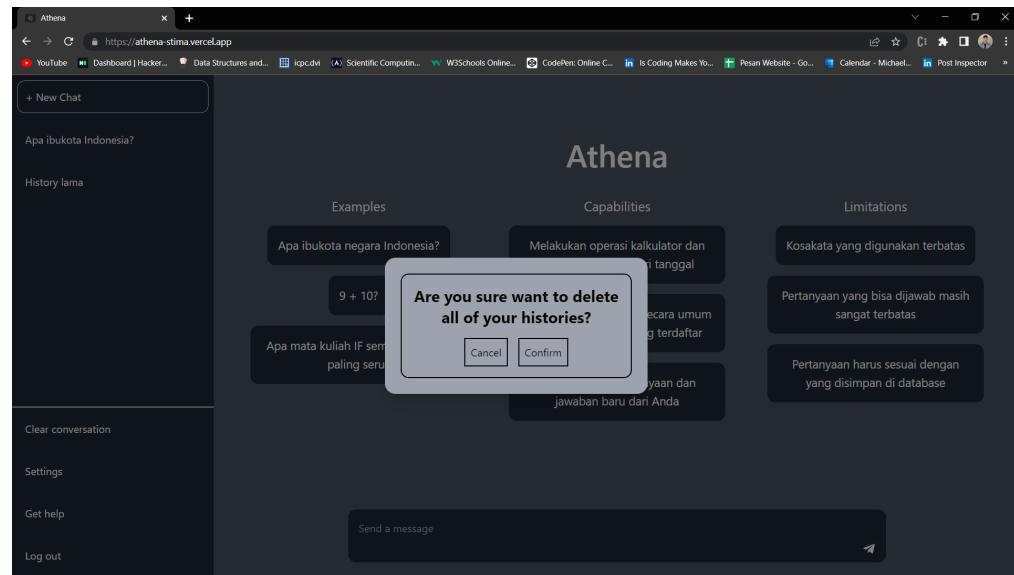
**Gambar 3.2.1.13** Menghapus Sesi/*History* dalam Desktop View



**Gambar 3.2.1.14** Menghapus Sesi/*History* dalam Mobile View

## 8. Menghapus seluruh sesi/history

Pengguna dapat menghapus seluruh sesi/history secara langsung dengan menekan tombol “Clear conversation” lalu menekan tombol “Confirm” pada popup konfirmasi penghapusan.



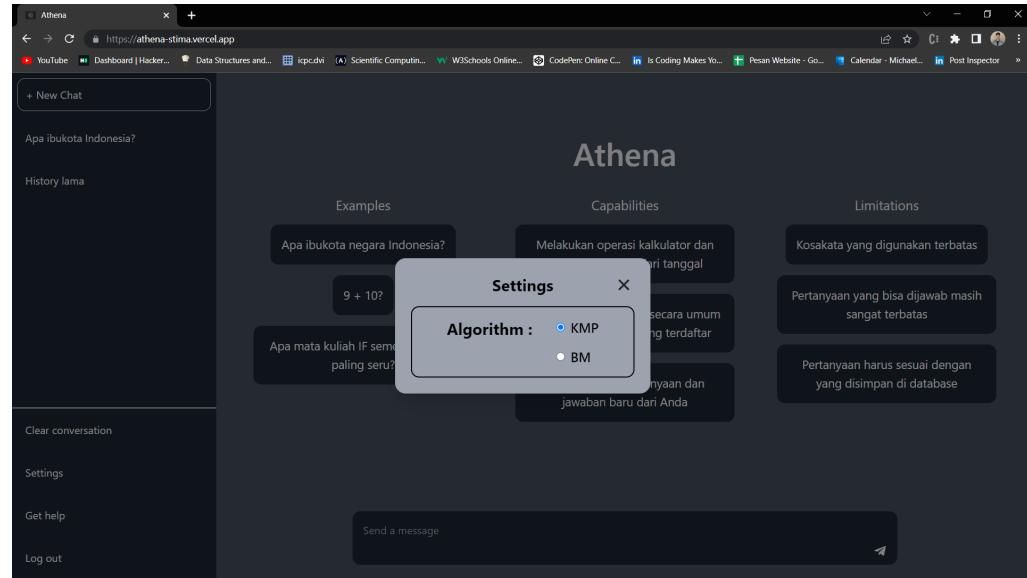
**Gambar 3.2.1.15** Menghapus Seluruh Sesi/*History* dalam Desktop View



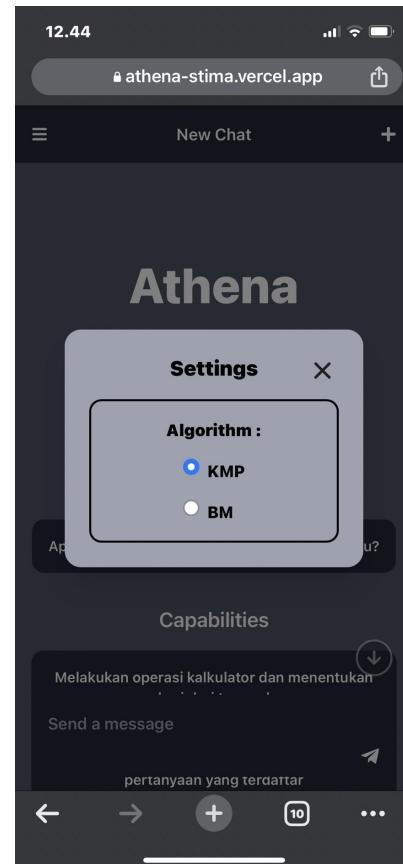
**Gambar 3.2.1.16** Menghapus Seluruh Sesi/*History* dalam Mobile View

9. Mengubah algoritma string matching

Pengguna dapat mengubah algoritma string matching yang digunakan dengan membuka popup settings dan memilih algoritma yang diinginkan.



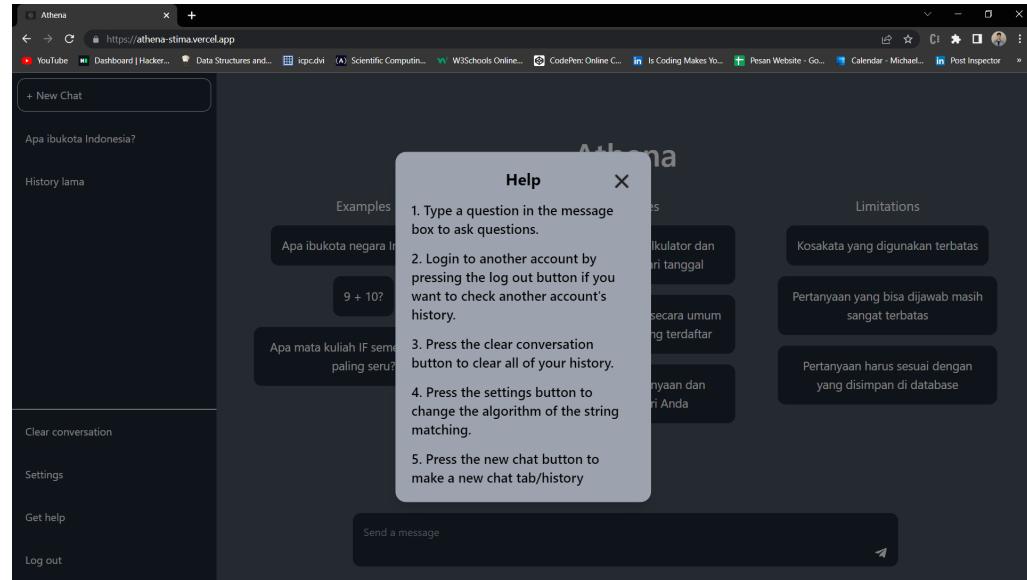
Gambar 3.2.1.17 Halaman *Settings* dalam Desktop View



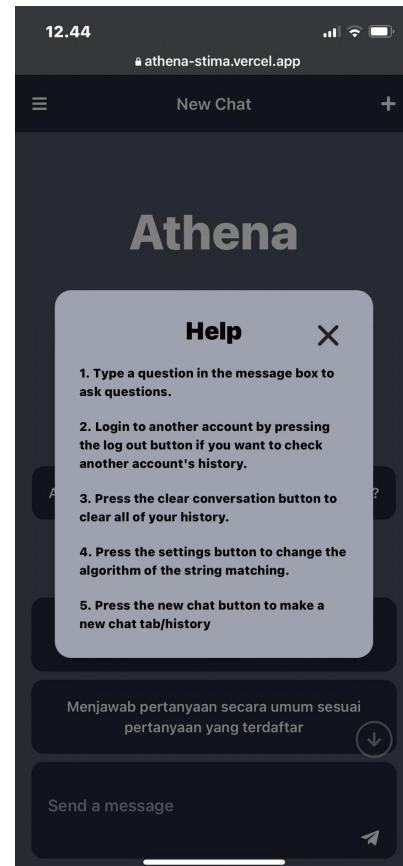
Gambar 3.2.1.18 Halaman *Settings* dalam Mobile View

## 10. Mendapatkan informasi bantuan penggunaan aplikasi

Pengguna dapat mengakses informasi bantuan dengan membuka popup help.



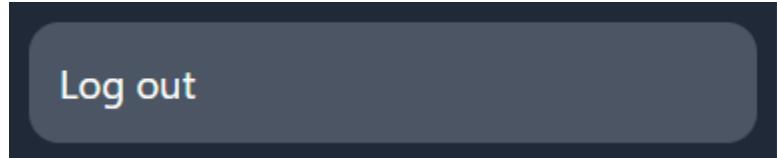
Gambar 3.2.1.19 Halaman *Help* dalam Desktop View



Gambar 3.2.1.20 Halaman *Help* dalam Mobile View

### 11. Tombol Logout

Untuk keluar dari aplikasi dan ganti akun, terdapat tombol logout di bawah kiri pada sidebar untuk logout.



**Gambar 3.2.1.21** Tombol Logout

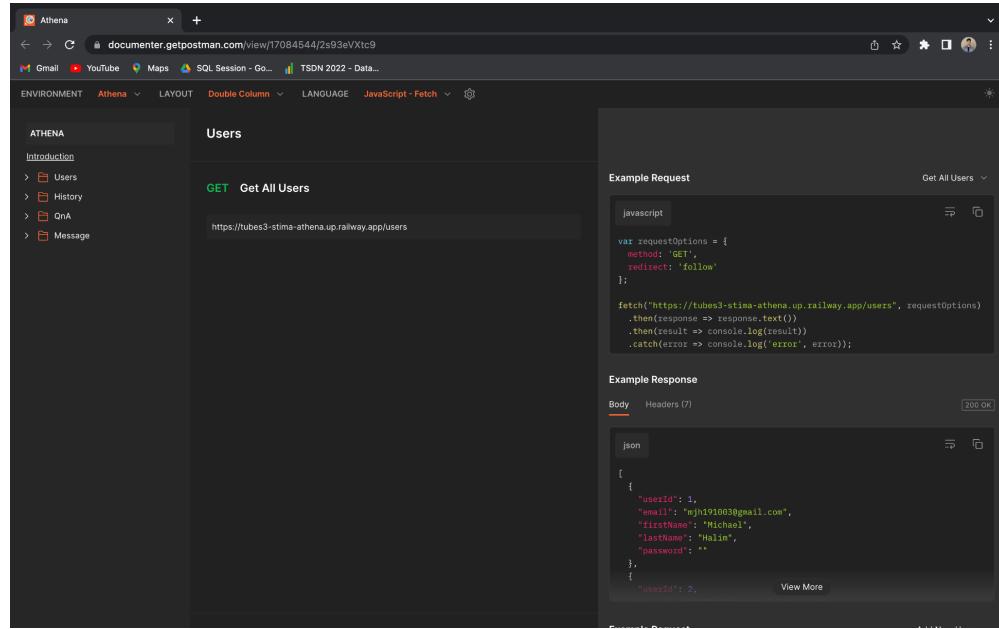
## 3.2.2 Back-end

Untuk arsitektur aplikasi web secara back-end, terdapat 5 directory utama, yaitu prisma (skema prisma dan kumpulan migrations), algorithms (kumpulan algoritma) yang terdiri dari Classification, StringMatching, dan StringSimilarity, controllers (kumpulan controller yang berfungsi sebagai jembatan antara service dan route), routes (kumpulan definisi endpoint/rute untuk diakses oleh front-end), dan services (kumpulan service yang berfungsi untuk memproses request untuk mengolah data langsung dengan database).

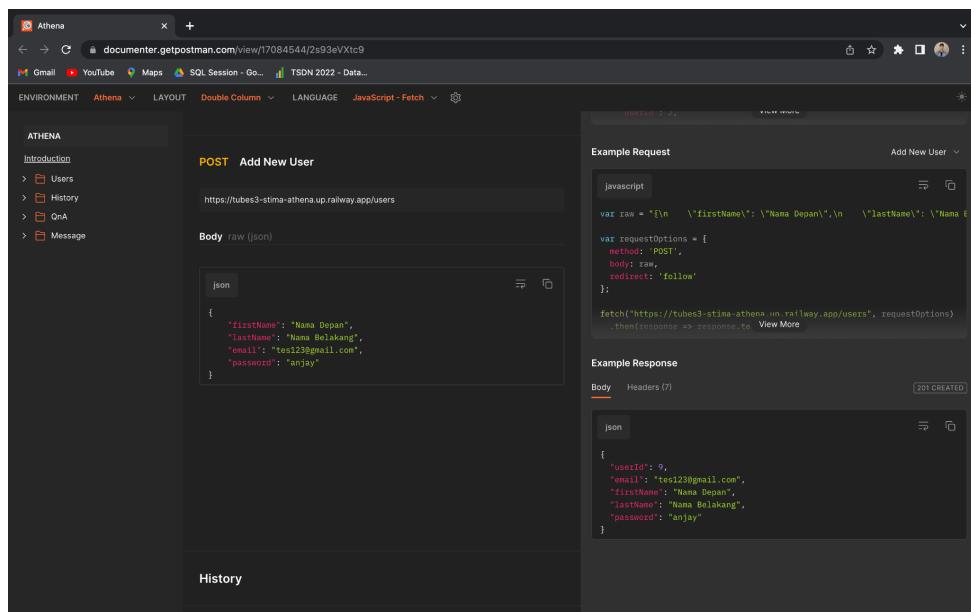
Berikut adalah kumpulan endpoint yang dibuat untuk memenuhi kebutuhan aplikasi.

### 1. Users

Untuk mengelola users, terdapat dua endpoint yang dibuat yaitu get all users (mendapatkan seluruh informasi semua users) dan post user (tambah user baru).



**Gambar 3.2.2.1** Dokumentasi Endpoint Get All Users



**Gambar 3.2.2.2** Dokumentasi Endpoint Post Add New User

## 2. QnA

Untuk mengelola QnA, terdapat empat endpoint yang dibuat, yaitu get all QnA (mendapatkan seluruh QnA), post create new QnA (membuat QnA baru), update QnA (memperbarui QnA yang sudah ada), dan delete QnA (menghapus QnA).

The screenshot shows the Postman interface for the 'QnA' section of the 'ATHENA' environment. On the left sidebar, under 'ATHENA', there is a 'QnA' category which is expanded, showing 'Introduction', 'Users', 'History', 'QnA', and 'Message'. The main panel displays the 'QnA' section with a 'GET Get All QnA' request. The 'Example Request' tab shows a JavaScript code snippet for making a GET request to the QnA endpoint. The 'Example Response' tab shows a JSON response containing two QnA entries, each with a question and an answer. One entry is fully visible, while the second is partially visible with a 'View More' link.

**Gambar 3.2.2.3 Dokumentasi Endpoint Get All QnA**

The screenshot shows the Postman interface for the 'QnA' section of the 'ATHENA' environment. On the left sidebar, under 'ATHENA', there is a 'QnA' category which is expanded, showing 'Introduction', 'Users', 'History', 'QnA', and 'Message'. The main panel displays the 'QnA' section with a 'POST Create New QnA' request. The 'Example Request' tab shows a JavaScript code snippet for making a POST request to the QnA endpoint with a raw JSON body. The 'Example Response' tab shows a JSON response indicating a successful creation with a status of 201 CREATED. Below this, there is another 'PUT Update QnA' request shown, with its URL being https://tubes3-stima-athena.up.railway.app/qna/{qna\_id}.

**Gambar 3.2.2.4 Dokumentasi Endpoint Post Create New QnA**

The screenshot shows the Postman interface for the 'ATHENA' environment. On the left, a sidebar lists 'Introduction', 'Users', 'History', 'QnA', and 'Message'. The main area displays the 'PUT Update QnA' endpoint. The URL is [https://tubes3-stima-athena.up.railway.app/qna/{qna\\_id}](https://tubes3-stima-athena.up.railway.app/qna/{qna_id}). The 'Body raw (json)' section contains the following JSON:

```
{
  "question": "Apakah Stima melelahkan?",
  "answer": "Oh tentu sangat tidak :)"
}
```

The 'Example Request' section shows the JavaScript code to make a PUT request:

```
var raw = '{\n  \"question\": \"Apakah Stima melelahkan?\", \n  \"answer\": \"Oh tentu sangat tidak :)\"\n}';

var requestOptions = {\n  method: 'PUT',\n  body: raw,\n  redirect: 'follow'\n};

fetch(`https://tubes3-stima-athena.up.railway.app/qna/37`, requestOptions)\n  .then(response => response.text()\n    .then(result => console.log(result))\n  )\n  .catch(error => console.log('error', error));
```

The 'Example Response' section shows the JSON response:

```
{
  "qnaId": 37,
  "question": "Apakah Stima melelahkan?",
  "answer": "Oh tentu sangat tidak :)"
}
```

**Gambar 3.2.2.5 Dokumentasi Endpoint Update QnA**

The screenshot shows the Postman interface for the 'ATHENA' environment. On the left, a sidebar lists 'Introduction', 'Users', 'History', 'QnA', and 'Message'. The main area displays the 'DELETE Delete QnA' endpoint. The URL is [https://tubes3-stima-athena.up.railway.app/qna/{qna\\_id}](https://tubes3-stima-athena.up.railway.app/qna/{qna_id}). The 'Body raw (json)' section contains the following JSON:

```
{
  "question": "Apakah Stima melelahkan?",
  "answer": "Oh tentu sangat tidak :)"
}
```

The 'Example Request' section shows the JavaScript code to make a DELETE request:

```
var requestOptions = {\n  method: 'DELETE',\n  redirect: 'follow'\n};

fetch(`https://tubes3-stima-athena.up.railway.app/qna/37`, requestOptions)\n  .then(response => response.text()\n    .then(result => console.log(result))\n  )\n  .catch(error => console.log('error', error));
```

The 'Example Response' section shows the response message:

No response body  
This request doesn't return any response body

**Gambar 3.2.2.6 Dokumentasi Endpoint Delete QnA**

### 3. History

Untuk mengelola history-history user, dibuatkan enam endpoint yaitu get all histories (mendapatkan seluruh history yang ada), get all histories by user ID (mendapatkan seluruh history berdasarkan user ID), post create new history

(membuat history baru untuk suatu user), delete all history by user ID (menghapus seluruh history milik sebuah user), delete history by user and history ID (menghapus suatu history milik suatu user).

The screenshot shows the Postman application interface for the ATHENA API. The left sidebar has a tree view with 'ATHENA' selected, under which 'History' is expanded, showing 'Get All Histories'. The main area shows a 'GET Get All Histories' request. The URL is <https://tubes3-stima-athena.up.railway.app/history>. The 'Example Request' tab contains a JavaScript code snippet for making the GET request. The 'Example Response' tab shows a JSON response with two history items:

```

[{"historyId": 1, "topic": "Contoh Topik", "userId": 1}, {"historyId": 1, "topic": "History lama", "userId": 8}]
  
```

**Gambar 3.2.2.7 Dokumentasi Endpoint Get All Histories**

The screenshot shows the Postman application interface for the ATHENA API. The left sidebar has a tree view with 'ATHENA' selected, under which 'History' is expanded, showing 'Get All History by User ID'. The main area shows a 'GET Get All History by User ID' request. The URL is <https://tubes3-stima-athena.up.railway.app/history/1>. The 'Example Request' tab contains a JavaScript code snippet for making the GET request. The 'Example Response' tab shows a JSON response with one history item:

```

[{"historyId": 1, "topic": "Anjay", "userId": 1}]
  
```

**Gambar 3.2.2.8 Dokumentasi Endpoint Get All Histories By User ID**

The screenshot shows the Postman interface with the 'ATHENA' environment selected. On the left, a sidebar lists 'ATHENA' categories: Introduction, Users, History, QnA, and Message. The main area displays a 'POST Create New History' endpoint at <https://tubes3-stima-athena.up.railway.app/history/>. The 'Example Request' tab shows a JavaScript code snippet for creating a history entry:

```
javascript
var raw = "";

var requestOptions = {
  method: 'POST',
  body: raw,
  redirect: 'follow'
};

fetch("https://tubes3-stima-athena.up.railway.app/history/{user_id}", requestOptions)
  .then(response => response.json())
  .then(result => console.log(result))
  .catch(error => console.error(error));

```

The 'Example Response' tab shows a JSON response with fields: historyId (18), topic (History), and userId (1).

**Gambar 3.2.2.9 Dokumentasi Endpoint Post Create New History**

The screenshot shows the Postman interface with the 'ATHENA' environment selected. On the left, a sidebar lists 'ATHENA' categories: Introduction, Users, History, QnA, and Message. The main area displays a 'DELETE Delete All History by User ID' endpoint at [https://tubes3-stima-athena.up.railway.app/history/{user\\_id}](https://tubes3-stima-athena.up.railway.app/history/{user_id}). The 'Example Request' tab shows a JavaScript code snippet for deleting all history entries for a user:

```
javascript
var requestOptions = {
  method: 'DELETE',
  redirect: 'follow'
};

fetch("https://tubes3-stima-athena.up.railway.app/history/1", requestOptions)
  .then(response => response.text())
  .then(result => console.log(result))
  .catch(error => console.error(error));

```

The 'Example Response' tab shows a 'No response body' message, indicating that this request does not return any response body.

**Gambar 3.2.2.10 Dokumentasi Endpoint Delete All History by User ID**

**ATHENA**

- Introduction
- > Users
- > History
  - DELETE Delete History by User ID and History ID
  - DELETE Delete All History by User ID
- > QnA
- > Message

**History**

**DELETE Delete History by User ID and History ID**

https://tubes3-stima-athena.up.railway.app/history/{user\_id}/{history\_id}

**Example Request**

```
javascript
var requestOptions = {
  method: 'DELETE',
  redirect: 'follow'
};

fetch("https://tubes3-stima-athena.up.railway.app/history/2/1", requestOptions)
  .then(response => response.text())
  .then(result => console.log(result))
  .catch(error => console.log(error));

```

**Example Response**

Body Headers (4) 204 NO CONTENT

No response body  
This request doesn't return any response body

**DELETE Delete All History by User ID**

**Example Request**

javascript

**Example Response**

204 NO CONTENT

**Gambar 3.2.2.11** Dokumentasi Endpoint Delete History by User and History ID

**ATHENA**

- Introduction
- > Users
- > History
  - DELETE Delete History by User ID and History ID
  - DELETE Delete All History by User ID
  - PUT Create New History
  - GET Get All Histories
  - GET Get All History by User ID
  - PUT Update History Topic
- > QnA
- > Message

**PUT Update History Topic**

https://tubes3-stima-athena.up.railway.app/history/1/1

**Body** raw (json)

```
json
{
  "topic": "Anjay"
}
```

**Example Request**

javascript

```
var raw = "\n  \"topic\": \"Anjay\"\n";  
  
var requestOptions = {  
  method: 'PUT',  
  body: raw,  
  redirect: 'follow'  
};  
  
fetch("https://tubes3-stima-athena.up.railway.app/history/1/1", requestOptions)
  .then(response => response.text())
  .then(result => console.log(result))
  .catch(error => console.log(error));

```

**Example Response**

Body Headers (7) 200 OK

```
json
[  
  {  
    "historyId": 1,  
    "topic": "Anjay",  
    "userId": 1  
  }  
]
```

**Gambar 3.2.2.12** Dokumentasi Endpoint Put Update History Topic

#### 4. Message

Untuk mengelola pesan-pesan, dibuatkan dua endpoint untuk mendapatkan seluruh pesan dalam suatu history oleh suatu user dan menambahkan pesan dari user di suatu history.

The screenshot shows the Postman interface for the 'ATHENA' environment. On the left sidebar, under the 'Message' category, there is a 'GET Get All Messages by User ID and History ID' request. The URL is listed as `https://tubes3-stima-athena.up.railway.app/message/{user_id}/{history_id}`. The 'Example Request' tab displays a JavaScript code snippet for making a GET request to the specified URL. The 'Example Response' tab shows a JSON array of messages, with one message partially visible:

```

[
  {
    "messageId": 1,
    "botMessage": "afrika.",
    "userMessage": "Apa ibukota nigeria?",
    "messageTimeStamp": "2023-05-03T14:34:01.987Z",
    "historyId": 1,
    "userId": 3
  }
]

```

**Gambar 3.2.2.13** Dokumentasi Endpoint Get All Messages by User and History ID

The screenshot shows the Postman interface for the 'ATHENA' environment. Under the 'Message' category, there is a 'POST Add Message' request. The URL is listed as `https://tubes3-stima-athena.up.railway.app/message/{user_id}/{history_id}?algo=kmp/bm`. The 'PARAMS' section shows a parameter named 'algo' with the value '(kmp/bm)'. The 'Body' section is set to 'raw (json)' and contains the following JSON payload:

```

{
  "userMessage": "Anjay"
}

```

The 'Example Request' tab displays a JavaScript code snippet for making a POST request to the specified URL with the provided JSON body. The 'Example Response' tab shows a JSON message, with one message partially visible:

```

[
  {
    "messageId": 1,
    "botMessage": "Tidak ada pertanyaan yang cocok dengan database!",
    "userMessage": "Anjay",
    "messageTimeStamp": "2023-05-04T08:09:03.666Z",
    "historyId": 1,
    "userId": 1
  }
]

```

**Gambar 3.2.2.14** Dokumentasi Endpoint Post Add Message

Untuk dokumentasi API dapat dilihat pada link berikut :  
<https://documenter.getpostman.com/view/17084544/2s93eVXtc9>

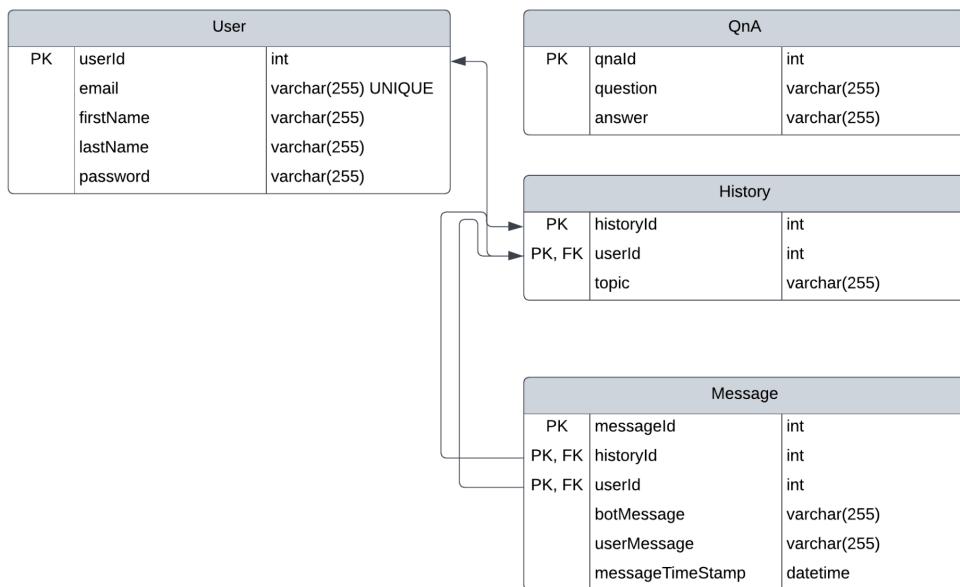
## BAB IV

# IMPLEMENTASI DAN PENGUJIAN

### 4.1. Spesifikasi Teknis Program

#### 4.1.1 Struktur Data

Karena aplikasi dibuat dengan database, maka diperlukan desain relasional model untuk penentuan bentuk data yang akan dipakai dan diolah dalam aplikasi. Berikut adalah hasil desain relasional model yang dipakai pada aplikasi kami.



**Gambar 4.1.1.1 Relational Model Database Aplikasi**

Oleh karena itu, representasi data dalam interface dibuat sebagai berikut.

```
interface History {  
    historyId: number;  
    userId: number;  
    topic: string;  
}  
  
interface Message {  
    messageId: number;  
    historyId: number;  
    userMessage: string;
```

```
    botMessage: string;
    timestamp: Date;
    userId: number;
}
```

```
interface QnA {
    qnaId: number;
    question: string;
    answer: string;
}
```

```
interface User {
    userId: number;
    email: string;
    firstName: string;
    lastName: string;
    password: string;
}
```

Untuk algoritma yang digunakan dalam backend, dibuat secara modular dengan membuat kelas-kelas sesuai fungsionalitasnya. Oleh karena itu, dibuat tiga kelas algoritma, yaitu

```
class Classification {
    isEmpty(s: string);
    isCalculator(s: string);
    priorityOps(ops1: String, ops2: String);
    calculate(ops: String, b: number, a: number);
    isDate(s: string);
    isDelete(s: string);
    isAdd(s: string);
}
```

```
class StringMatching {
    BMAlgorithm(pattern : string, text : string);
    badMatch(pattern : string, patternLength : number, badMatchTable
: Array<number>);
    KMPAlgorithm(pattern: string, text: string);
    lpsArray(pattern: string);
}
```

```
class StringSimilarity {
```

```

    similarity(s1: string, s2: string);
    editDistance(s1: string, s2: string);
}

```

#### 4.1.2 Fungsi dan Prosedur

Untuk fungsi dan prosedur yang diimplementasikan dalam front-end adalah sebagai berikut.

<code>handleLogin(e: FormEvent) → Handle process login</code>
<pre> const handleLogin = async (e: FormEvent) =&gt; {   e.preventDefault();    try {     if (email === ""    password === "") throw new Error("Gagal Login");      // Get all users     const response = await fetch(`process.env.NEXT_PUBLIC_API_URL}/users`, {       method: "GET",     });     const data = await response.json();      for (let user of data) {       if (user.email === email &amp;&amp; user.password === password) {         setId(user.id);         setAuthenticated(true);         toast.success("Berhasil Login");         return;       }     }      throw new Error("Gagal Login");   } catch (e) {     toast.error("Gagal Login. Pastikan semua input terisi dan sesuai!");   } }; </pre>
<code>handleRegister(e: FormEvent) → Handle process register</code>

<pre> const handleRegister = async (e: FormEvent) =&gt; {   e.preventDefault();    try {     if (       email === ""          password === ""          firstName === ""    </pre>
---

```
lastName === ""
)
throw new Error("Gagal Register");

const body = {
  email: email,
  password: password,
  firstName: firstName,
  lastName: lastName,
};

await fetch(` ${process.env.NEXT_PUBLIC_API_URL}/users` , {
  method: "POST",
  body: JSON.stringify(body),
  headers: { "Content-Type": "application/json" },
}).then((response) => {
  if (response.ok) {
    // Do Nothing
  } else {
    throw new Error(response.statusText);
  }
});

toast.success("Berhasil Register");
setRegister(false);
} catch (e) {
  toast.error("Gagal Register! Pastikan semua input diisi!")
}
};
```

`searchTopic(historyId: number) → Return topic of a history ID`

```
const searchTopic = (historyId: number): string => {
    for (let el of history) {
        if (el.historyId === historyId) {
            return el.topic;
        }
    }
    // Not found
    return "";
};
```

```
submitQuestion(e: FormEvent) → Handle submit user message
```

```
const submitQuestion = async (e: FormEvent) => {
  e.preventDefault();
  if (question !== "") {
    const body = {
      userMessage: question,
    };
    try {
```

```

    const data = await fetch(`

${process.env.NEXT_PUBLIC_API_URL}/message/${userId}/${selectedHistory}?algo=${algorithm}`,
{
  method: "POST",
  body: JSON.stringify(body),
  headers: { "Content-Type": "application/json" },
}
).then((response) => {
  if (response.ok) {
    return response.json();
  } else {
    throw new Error(response.statusText);
  }
});

await data.sort((a: Message, b: Message) => a.messageId - b.messageId);
setMessage(data);
setAfterAsk(true);

if (selectedHistory === 0) {
  setLoadingHistory(true);
  setIsNewHistory(true);
  setTriggerGetHistories(!triggerGetHistories);
}
setSelectedHistory(data[0].historyId);
setQuestion("");
setLoading(false);
} catch (e) {
  toast.error("Gagal tambah pertanyaan!");
  setLoading(false);
}
} else {
  setLoading(false);
}
});

```

#### deleteQuestion(e: FormEvent) → Handle delete question

```

const deleteHistories = async (e: FormEvent) => {
  e.preventDefault();
  try {
    await fetch(` ${process.env.NEXT_PUBLIC_API_URL}/history/${userId}`, {
      method: "DELETE",
    }).then((response) => {
      if (response.ok) {
        // Do Nothing
      } else {
        throw new Error(response.statusText);
      }
    });
  
```

```
        setTriggerGetHistories(!triggerGetHistories);
        setMessage([]);
        setHistory([]);
        setSelectedHistory(0);
        setQuestion("");
        toast.success("Berhasil delete seluruh history!");
        setLoadingHistory(false);
    } catch (e) {
        toast.error("Gagal delete seluruh history!");
        setLoadingHistory(false);
    }
};
```

#### deleteHistory(e: FormEvent, historyId: number) → Handle delete history

```
const deleteHistory = async (e: FormEvent, historyId: number) => {
    e.preventDefault();
    try {
        await fetch(
            `${process.env.NEXT_PUBLIC_API_URL}/history/${userId}/${historyId}`,
            {
                method: "DELETE",
            }
        ).then((response) => {
            if (response.ok) {
                // Do Nothing
            } else {
                throw new Error(response.statusText);
            }
        });
        setTriggerGetHistories(!triggerGetHistories);
        if (selectedHistory === historyId) {
            setMessage([]);
            setHistory([]);
            setSelectedHistory(0);
            setQuestion("");
        }
        toast.success(`Berhasil delete history!`);
        setLoadingHistory(false);
    } catch (e) {
        toast.error(`Gagal delete history!`);
        setLoadingHistory(false);
    }
};
```

#### updateHistory(e: FormEvent, historyId: number) → Handle update history

```
const updateHistory = async (e: FormEvent, historyId: number) => {
    e.preventDefault();
```

```

try {
  const body = {
    topic: newTopic,
  };

  const result = await fetch(
    `${process.env.NEXT_PUBLIC_API_URL}/history/${userId}/${historyId}`,
    {
      method: "PUT",
      body: JSON.stringify(body),
      headers: { "Content-Type": "application/json" },
    }
  ).then((response) => {
    if (response.ok) {
      return response.json();
    } else {
      throw new Error(response.statusText);
    }
  });
}

result.sort(function (a: History, b: History) {
  return b.historyId - a.historyId;
});

setHistory(result.slice(0, 10));
toast.success(`Berhasil update history!`);
 setLoadingHistory(false);
} catch (e) {
  toast.error(`Gagal update history!`);
  setLoadingHistory(false);
}
};

```

Untuk fungsi dan prosedur yang diimplementasikan dalam back-end adalah sebagai berikut.

### Classification.ts

`isEmpty(s : string) → Check if string is empty or not`

```

// Classify is empty or not
const regexEmpty = /^$\s*$/;
let flag;
if (regexEmpty.test(s)){
  flag = true;
} else {
  flag = false;
}
return flag;

```

```
isCalculator(s : string) → Check and handle calculator feature
```

```
// Classify is calculator or not
const regexCalculator=
/^(?:(:Berapa|Hitung|Kalkulasi)\s+)?([()d+\-*/.^s]+)(\?)?$/i;
const operatorsArray = ["+", "-", "*", "/", "^"];
const match = regexCalculator.exec(s);
let result = 0;
let flag = false;

if (match) {
    let values = [];
    let operators = [];
    let input = match[1].split('');

    for (let i = 0; i < input.length; i++){
        if (input[i] == ' '){
            // Checking if before and after number or not
            let j = i, k = i;
            let wrong = false;
            let done = false;
            while(--j >= 0 && !done){
                if(input[j] >= '0' && input[j] <= '9'){
                    while(++k < input.length && !wrong){
                        if(input[k] >= '0' && input[k] <= '9'){
                            done = true;
                            wrong = true;
                            break;
                        } else if(input[k] != ' '){
                            done = true;
                            break;
                        }
                    }
                } else if(input[j] != ' '){
                    done = true;
                    break;
                }
            }
            if(!wrong){
                continue;
            } else {
                flag = true;
                return {flag, undefined};
            }
        }
        if(input[i] >= '0' && input[i] <= '9'){
            let tempValue = input[i];

            while (i+1 < input.length && ((input[i+1] >= '0' && input[i+1] <= '9') ||
input[i+1] == '.')){
                tempValue += input[++i];
            }

            values.push(Number(tempValue));
        }
        if (input[i] == '('){
            if (input[i+1] == '-'){
                let j = i+1;
                let wrong = false;
                let done = false;
                while(j < input.length && !done){
                    if(input[j] >= '0' && input[j] <= '9'){
                        while(j+1 < input.length && !wrong){
                            if(input[j+1] >= '0' && input[j+1] <= '9'){
                                done = true;
                                wrong = true;
                                break;
                            } else if(input[j+1] == ')'){
                                done = true;
                                break;
                            }
                        }
                    } else if(input[j] != ' '){
                        done = true;
                        break;
                    }
                }
                if(!wrong){
                    continue;
                } else {
                    flag = true;
                    return {flag, undefined};
                }
            }
        }
    }
}
```

```

        values.push(0);
    }
    operators.push(input[i]);
}

if (input[i] == ')'){
    while (operators[operators.length - 1] != '('){
        const tempCalOps = operators.pop();
        const tempCalVal1 = values.pop() as number;
        const tempCalVal2 = values.pop() as number;
        if (tempCalOps != undefined && tempCalVal1 != undefined &&
tempCalVal2 != undefined){
            values.push(calculate(tempCalOps, tempCalVal1, tempCalVal2));
        }
    }
    operators.pop();
}

if (operatorsArray.includes(input[i])){
    if(input[i] == "-" && i == 0){
        values.push(0);
    }
    let tempOps = input[i];
    while (operators.length > 0 && priorityOps(tempOps,
operators[operators.length -1])){
        const tempCalOps = operators.pop();
        const tempCalVal1 = values.pop() as number;
        const tempCalVal2 = values.pop() as number;
        if (tempCalOps != undefined && tempCalVal1 != undefined &&
tempCalVal2 != undefined){
            values.push(calculate(tempCalOps, tempCalVal1, tempCalVal2));
        }
    }
    operators.push(tempOps);
}
}

while (operators.length > 0) {
    const tempCalOps = operators.pop();
    const tempCalVal1 = values.pop() as number;
    const tempCalVal2 = values.pop() as number;
    if (tempCalOps != undefined && tempCalVal1 != undefined && tempCalVal2 !=
undefined){
        values.push(calculate(tempCalOps, tempCalVal1, tempCalVal2));
    }
}

result = values.pop() as number;
flag = true;
return {flag, result};
} else {
    return {flag, result} ;
}

function priorityOps (ops1: String, ops2: String) {
    if (ops2 == '(' || ops2 == ')' || ops1 == "^"){
        return false;
    }
    else if ((ops1 == '*' || ops1 == '/') && (ops2 == "+" || ops2 == "-")){
        return false;
    }
}

```

```

        }
    else {
        return true;
    }
}

function calculate(ops: String, b: number, a: number){
    switch (ops){
        case '+':
            return a + b;
        case '-':
            return a - b;
        case '*':
            return a * b;
        case '/':
            return a / b;
        case "^":
            return a ** b;
    }
    return 0;
}

```

### isDate(s : string) → Check and handle date feature

```

// Classify is date or not
const inputCleaned = s.replace(/(?:\?|\?)*$/g, "").replace(/\s+$/, "");
const inputArr = inputCleaned.split(" ");
const dateString = inputArr[inputArr.length - 1];
const dateRegex = /^(\d{1,2})/(\d{1,2})/(\d{4})$/;
let match = dateString.match(dateRegex);
let flag = true;
let result = 0;

if (!match){
    flag = false;
} else {
    let year = parseInt(match[3]);
    let month = parseInt(match[2]);
    let day = parseInt(match[1]);
    let maxDay = 0;

    // set max day
    if (year >= 1753){
        if (month == 2) {
            if (year % 4 == 0 && year % 100 != 0){
                maxDay = 29;
            } else {
                maxDay = 28;
            }
        } else if (month == 4 || month == 6 || month == 9 || month == 11){
            maxDay = 30;
        } else {
            maxDay = 31;
        }

        if (day > maxDay || day < 1 || month < 1 || month > 12){
            flag = false;
        } else {

```

```

// adjust month and year to Zeller's Rule
if (month < 3) {
    month +=12;
    year--;
}
if(month >=3) {
    month -=2;
}

const k = day;
const m = month;
const D = year % 100;
const C = Math.floor( year / 100);

// Zeller's Rule
result = k + Math.floor((13 * m -1) / 5) + D + Math.floor(D / 4) +
Math.floor(C / 4) - 2 * C;
result = Math.abs(result % 7);
}
}
}

return { flag, result };

```

**isDelete(s : string) → Check and handle delete feature**

```

// Classify is delete or not
const regexDelete: RegExp = /^Hapus pertanyaan .+$/i;
if (regexDelete.test(s)) {
    return true;
} else {
    return false;
}

```

**isAdd(s : string) → Check and handle add feature**

```

// Classify is add or not
const regexAdd: RegExp = /^Tambahkan pertanyaan .+ dengan jawaban .+$/i;
if (regexAdd.test(s)) {
    return true;
} else {
    return false;
}

```

## StringMatching.ts

**BMAlgorithm(pattern : string, text : string) → Main BM Algorithm**

```

// Convert the pattern and text to lowercase
let patternFix = pattern.toLowerCase();
let textFix = text.toLowerCase();

```

```

// Get the lengths of the pattern and text
let patternLength = patternFix.length;
let textLength = textFix.length;

// Create an array to store the bad match table
let badMatchTable = new Array(patternLength);

// Initialize variables
let shift = 0;
let idxPattern;
let flag = false;

// Create the bad match table
this.badMatch(patternFix, patternLength, badMatchTable);

// Check if the pattern is shorter than the text
if (patternLength <= textLength){
    // Loop through the text
    while ((textLength - patternLength) >= shift){
        idxPattern = patternLength - 1;

        while (idxPattern >= 0 && patternFix.charAt(idxPattern) ==
textFix.charAt(shift+idxPattern)){
            idxPattern--;
        }

        if (idxPattern < 0){
            flag = true;
            if (shift + patternLength < textLength){
                shift += textLength - badMatchTable[text.charCodeAt(shift +
patternLength).charCodeAt(0)];
            }
            else {
                shift += 1;
            }
        }
        else {
            flag = false;
            shift += Math.max(1, idxPattern -
badMatchTable[text.charCodeAt(shift+idxPattern).charCodeAt(0)]);
        }
    }
} else {
    flag = false;
}
return flag;

```

`badMatch(pattern : string, patternLength : number,  
badMatchTable : Array<number>) → Creates bad match table`

```

// Initialize the bad match table
for(let i = 0; i < 256; i++){
    badMatchTable[i] = -1;
}

// Fill in the bad match table with the appropriate values
for(let j = 0; j < patternLength; j++){
    badMatchTable[pattern.charCodeAt(j)] = j;
}

```

### KMPAlgorithm(pattern: string, text: string) → Main KMP Algorithm

```

// Convert the pattern and text to lowercase
let patternFix = pattern.toLowerCase();
let textFix = text.toLowerCase();

// Get the lengths of the pattern and text
let patternLength = patternFix.length;
let textLength = textFix.length;

// Create an array to store the bad match table
let patternLPS = this.lpsArray(patternFix);
let flag = false;

// Initialize variables
let i = 0;
let j = -1;

// Check if the pattern is shorter than the text
if (patternLength <= textLength){
    // Loop through the text
    while (i < textLength && !flag) {
        if (patternFix.charAt(j + 1) == textFix.charAt(i)) {
            if (j + 1 == patternLength - 1) {
                flag = true;
            }
            j++;
            i++;
        } else {
            if (j == -1) {
                i++;
            } else {
                j = patternLPS[j + 1] - 1;
            }
        }
    }
}
return flag;

```

`lpsArray(pattern: string) → generate the Longest Prefix Suffix (LPS) array of a given pattern`

```
let patternLength = pattern.length;
let flag = false;
let backTo = 0;
const patternChar = [];
const lpsArray = [];

for (let i = 0; i < patternLength; i++) {
    patternChar[i] = pattern.charAt(i);
}

for (let j = 0; j <= patternLength; j++) {
    if (j == 0) {
        lpsArray[j] = -1;
        flag = false;
        backTo = 0;
    } else if (j == 1) {
        lpsArray[j] = 0;
        flag = false;
        backTo = 0;
    } else {
        for (let k = 0; k < j - 1; k++) {
            if (patternChar[j - 1] == patternChar[k]) {
                flag = true;
                backTo++;
                lpsArray[j] = backTo;
                break;
            } else {
                flag = false;
                lpsArray[j] = 0;
            }
        }
        if (!flag) {
            backTo = 0;
        }
    }
}
return lpsArray;
```

**StringSimilarity.ts**

`similarity(s1: string, s2: string) → Return percentage similarity`

```
let result;
```

```

// Check if either string is empty, set result to 1.0
if (s1.length == 0 || s2.length == 0) {
    result = 1.0;
} else if (s1.length > s2.length) {
    // Calculate similarity based on s1 being longer than s2
    result = (s1.length - this.editDistance(s1, s2)) / Number(s1.length);
} else {
    // Calculate similarity based on s2 being longer than s1
    result = (s2.length - this.editDistance(s1, s2)) / Number(s2.length);
}

return result;

```

**editDistance(s1: string, s2: string) → Calculates edit distance between two strings**

```

// Convert both strings to lowercase
let string1 = s1.toLowerCase();
let string2 = s2.toLowerCase();
const matrix = [];

// Init first column
for (let i = 0; i <= string2.length; i++) {
    matrix[i] = [i];
}

// Init first row
for (let j = 0; j <= string1.length; j++) {
    matrix[0][j] = j;
}

// Fill the rest of matrix from 3 matrix above left
for (let i = 1; i <= string2.length; i++) {
    for (let j = 1; j <= string1.length; j++) {
        if (string2.charAt(i - 1) == string1.charAt(j - 1)) {
            matrix[i][j] = matrix[i - 1][j - 1];
        } else {
            matrix[i][j] =
                Math.min(
                    matrix[i - 1][j - 1],
                    Math.min(matrix[i][j - 1], matrix[i - 1][j])
                ) + 1;
        }
    }
}

return matrix[string2.length][string1.length];

```

## Main Algorithm for Chat Response (message-service.ts)

```
async createMessageInUserHistory(
  data: any,
  userId: number,
  historyId: number,
  algorithm: string
): Promise<Message[]> {
  /* "We are implementing a 'manual' auto-increment
   * of the messageId because if it were auto-incrementing,
   * the auto-increment sequence would be preserved within different historyId.
   * This would make the messageId unique, allowing it to identify
   * a single record in the table. However, having a composite
   * primary key (i.e. historyId and messageId) is not good practice,
   * even though historyId alone can identify a single record.*/

  /* Currently, the 'manual' auto-increment implementation is not problematic
   * because it is not possible to delete an individual record in the message
   * table. However, it would cause issues if deleting an individual record were possible.
  */
  const userMessageCount = (
    await prisma.message.findMany({
      where: { historyId: historyId },
      select: { messageId: true },
    })
  ).length;

  // Classification Regex Algorithm
  const classify = new Classification();

  // String Matching Algorithm
  const stringMatching = new StringMatching();

  // String Similarity Algorithm
  const stringSimilarity = new StringSimilarity();

  // Main Algorithm
  var count: number = 0;
  let answer: string = "";
  const questions = data.userMessage.split("\n");

  for (let question of questions) {
    const qna = await prisma.qna.findMany();

    var idx = 0;
    var percentageArray = [];
    var resultDate = classify.isDate(question);
    var resultCalcu = classify.isCalculator(question);

    if (classify.isEmpty(question)) {
      answer += "Pertanyaan kosong.";
    } else if (resultDate.flag) {
      var theDate = resultDate.result;
      var days = [
        "Minggu",
        "Senin",
        "Selasa",
        "Rabu",
        "Kamis",
      ];
      var dayIndex = theDate % days.length;
      answer += `Tgl ${theDate} ${days[dayIndex]}.`;
    } else if (resultCalcu.flag) {
      var result = resultCalcu.result;
      answer += `Hasil perhitungan: ${result}`;
    }
  }
}
```

```

        "Jumat",
        "Sabtu",
    ];
var result = days[theDate];
answer += "Hari " + result + ".";
} else if (resultCalcu.flag) {
let expression = question;
try {
    if (expression[expression.length - 1] === "?") {
        expression = expression.slice(0, -1);
    }
    if (resultCalcu.result === undefined) {
        answer += "Sintaks persamaan tidak sesuai.";
    } else {
        if (resultCalcu.result.toString() === "Infinity") {
            answer += "Hasilnya adalah undefined.";
        } else {
            answer +=
                "Hasilnya adalah " + resultCalcu.result.toString() + ".";
        }
    }
} catch (e) {
    answer += "Sintaks persamaan tidak sesuai.";
}
} else if (classify.isDelete(question)) {
let regexQuestion = /^Hapus pertanyaan\s+(.*)$/i;
let deleteQuestion = question.match(regexQuestion) as RegExpMatchArray;
let found = false;
let qnaId = 0;
let questionArray = [];
let qIDArray = [];
let count = 0;
for (let el of qna) {
    if (algorithm === "kmp") {
        found = stringMatching.KMPAlgorithm(
            deleteQuestion[1].toLocaleLowerCase(),
            el.question.toLocaleLowerCase()
        );
    } else if (algorithm == "bm") {
        found = stringMatching.BMAlgorithm(
            deleteQuestion[1].toLocaleLowerCase(),
            el.question.toLocaleLowerCase()
        );
    }
    if (found) {
        questionArray.push(el.question.toLocaleLowerCase());
        qIDArray.push(el.qnaId);
        qnaId = el.qnaId;
        count++;
    }
}
if (count == 1) {
    let accuracy = stringSimilarity.similarity(
        deleteQuestion[1].toLocaleLowerCase(),
        questionArray[0]
    );
    if (accuracy >= 0.9) {
        qnaId = qIDArray[0];
        // Delete Pertanyaan
        const deleteQnA = new QnaService();

```

```

        await deleteQnA.deleteQna(qnaId);
        answer += "Pertanyaan " + questionArray[0] + " telah dihapus.";
    } else {
        answer +=
            "Tidak ada pertanyaan " + deleteQuestion[1] + " pada database!";
    }
} else if (count > 1) {
    let max = 0;
    let accuracy = 0;
    let i = 0;
    let idx = 0;
    let tempQuestion = "";
    for (let q of questionArray) {
        accuracy = 0;
        accuracy = stringSimilarity.similarity(
            deleteQuestion[1].toLocaleLowerCase(),
            q
        );
        if (accuracy > max) {
            max = accuracy;
            idx = i;
            tempQuestion = q;
        }
        i++;
    }
    if (max >= 0.9) {
        qnaId = qIDArray[idx];
        const deleteQnA = new QnaService();
        await deleteQnA.deleteQna(qnaId);
        answer += "Pertanyaan " + tempQuestion + " telah dihapus.";
    } else {
        answer +=
            "Tidak ada pertanyaan " + deleteQuestion[1] + " pada database!";
    }
} else {
    answer +=
        "Tidak ada pertanyaan " + deleteQuestion[1] + " pada database!";
}
} else if (classify.isAdd(question)) {
    let regexQuestion = /^Tambahkan pertanyaan (.+) dengan jawaban (.+)$/i;
    let addQuestion = question.match(regexQuestion) as RegExpMatchArray;
    let found = false;
    let qnaId = 0;
    let questionArray = [];
    let qIDArray = [];
    let count = 0;
    for (let el of qna) {
        if (algorithm === "kmp") {
            found = stringMatching.KMPAlgorithm(
                addQuestion[1].toLocaleLowerCase(),
                el.question.toLocaleLowerCase()
            );
        } else if (algorithm === "bm") {
            found = stringMatching.BMAlgorithm(
                addQuestion[1].toLocaleLowerCase(),
                el.question.toLocaleLowerCase()
            );
        }
        if (found) {
            questionArray.push(el.question.toLocaleLowerCase());
        }
    }
}

```

```

        qIDArray.push(el.qnaId);
        qnaId = el.qnaId;
        count++;
    }
}
if (count == 1) {
    let accuracy = stringSimilarity.similarity(
        addQuestion[1].toLocaleLowerCase(),
        questionArray[0]
    );
    if (accuracy >= 0.9) {
        qnaId = qIDArray[0];
        // Update jawaban pertanyaan
        const updateQnA = new QnaService();
        await updateQnA.updateQna(
            { question: questionArray[0], answer: addQuestion[2] },
            qnaId
        );
        answer += 
            "Pertanyaan " +
            questionArray[0] +
            " sudah ada! Jawaban di-update ke " +
            addQuestion[2] +
            ".";
    } else {
        // Tambahkan pertanyaan ke database
        const addQnA = new QnaService();
        await addQnA.createQna({
            question: addQuestion[1],
            answer: addQuestion[2],
        });
        answer += 
            "Pertanyaan " + addQuestion[1] + " telah ditambah ke database!";
    }
} else if (count > 1) {
    let max = 0;
    let accuracy = 0;
    let i = 0;
    let idx = 0;
    let tempQuestion = "";
    for (let q of questionArray) {
        accuracy = 0;
        accuracy = stringSimilarity.similarity(
            addQuestion[1].toLocaleLowerCase(),
            q
        );
        if (accuracy > max) {
            max = accuracy;
            idx = i;
            tempQuestion = q;
        }
        i++;
    }
    if (max >= 0.9) {
        qnaId = qIDArray[idx];
        const updateQnA = new QnaService();
        await updateQnA.updateQna(
            { question: tempQuestion, answer: addQuestion[2] },
            qnaId
        );
    }
}

```

```

        answer +=
            "Pertanyaan " +
            tempQuestion +
            " sudah ada! Jawaban di-update ke " +
            addQuestion[2] +
            ".";
    } else {
        // Tambahkan pertanyaan ke database
        const addQnA = new QnaService();
        await addQnA.createQna({
            question: addQuestion[1],
            answer: addQuestion[2],
        });
        answer +=
            "Pertanyaan " + addQuestion[1] + " telah ditambah ke database!";
    }
} else {
    // Tambahkan pertanyaan ke database
    const addQnA = new QnaService();
    await addQnA.createQna({
        question: addQuestion[1],
        answer: addQuestion[2],
    });
    answer +=
        "Pertanyaan " + addQuestion[1] + " telah ditambah ke database!";
}
} else {
    // Ask Question
    let found = false;
    let tempAnswer = "";
    let questionArray = [];
    let answerArray = [];
    let count = 0;
    for (let el of qna) {
        if (algorithm === "kmp") {
            found = stringMatching.KMPAlgorithm(
                question.toLocaleLowerCase(),
                el.question.toLocaleLowerCase()
            );
        } else if (algorithm == "bm") {
            found = stringMatching.BMAlgorithm(
                question.toLocaleLowerCase(),
                el.question.toLocaleLowerCase()
            );
        }
        if (found) {
            if (el.answer[el.answer.length - 1] == ".") {
                tempAnswer = el.answer;
            } else {
                tempAnswer = el.answer + ".";
            }
            questionArray.push(el.question.toLocaleLowerCase());
            answerArray.push(tempAnswer);
            count++;
        }
    }
    if (count == 1) {
        let accuracy = stringSimilarity.similarity(
            question.toLocaleLowerCase(),
            questionArray[0]

```

```

    );
    if (accuracy >= 0.9) {
        answer += answerArray[0];
    } else {
        count = 0;
    }
}
if (count == 0 || count != 1) {
    let percentage: number = 0.0;
    let finalAnswer: string = "";

    for (let el of qna) {
        let temp = stringSimilarity.similarity(
            question.toLocaleLowerCase(),
            el.question.toLocaleLowerCase()
        );
        var percentages = temp;
        var percentageQuestion = el.question;
        var percentageArrayData = { percentages, percentageQuestion };
        percentageArray[idx] = percentageArrayData;
        idx++;
        if (temp > percentage) {
            percentage = temp;
            finalAnswer = el.answer;
        }
    }
}

if (percentage >= 0.9) {
    if (finalAnswer[finalAnswer.length - 1] == ".") {
        answer += finalAnswer;
    } else {
        answer += finalAnswer + ".";
    }
} else if (percentage < 0.9 && percentage > 0.5) {
    answer +=
        "Pertanyaan tidak ditemukan di database.\nApakah maksud Anda:\n";
    percentageArray.sort((a, b) => b.percentages - a.percentages);
    if (qna.length < 3) {
        for (let i = 0; i < qna.length; i++) {
            if (i != qna.length - 1) {
                if (percentageArray[i + 1].percentages > 0.5) {
                    answer +=
                        (i + 1).toString() +
                        ". " +
                        percentageArray[i].percentageQuestion +
                        "\n";
                } else {
                    answer +=
                        (i + 1).toString() +
                        ". " +
                        percentageArray[i].percentageQuestion;
                    break;
                }
            } else {
                answer +=
                    (i + 1).toString() +
                    ". " +
                    percentageArray[i].percentageQuestion;
            }
        }
    }
}

```

```

    } else {
      for (let i = 0; i < 3; i++) {
        if (i != 2) {
          if (percentageArray[i + 1].percentages > 0.5) {
            answer += 
              (i + 1).toString() +
              ". " +
              percentageArray[i].percentageQuestion +
              "\n";
          } else {
            answer += 
              (i + 1).toString() +
              ". " +
              percentageArray[i].percentageQuestion;
            break;
          }
        } else {
          answer += 
            (i + 1).toString() +
            ". " +
            percentageArray[i].percentageQuestion;
        }
      }
    } else {
      answer += "Tidak ada pertanyaan yang cocok dengan database!";
    }
  }
}

if (count != questions.length - 1) {
  answer += "\n\n";
  count++;
}
}

if (historyId === 0) {
  // Create new history
  const createHistory = new HistoryService();

  const newHistory = await createHistory.createUserHistory(
    userId,
    data.userMessage
  );

  await prisma.message.create({
    data: {
      messageId: userMessageCount + 1,
      botMessage: answer,
      userMessage: data.userMessage,
      userId: userId,
      historyId: newHistory.historyId,
    },
  });
}

const allMessages = await this.getAllMessagesInUserHistory(
  userId,
  newHistory.historyId
);

return allMessages;

```

```

} else {
    await prisma.message.create({
        data: {
            messageId: userMessageCount + 1,
            botMessage: answer,
            userMessage: data.userMessage,
            userId: userId,
            historyId: historyId,
        },
    });
}

const allMessages = await this.getAllMessagesInUserHistory(
    userId,
    historyId
);

return allMessages;
}
}

```

## 4.2. Tata Cara Penggunaan Sistem

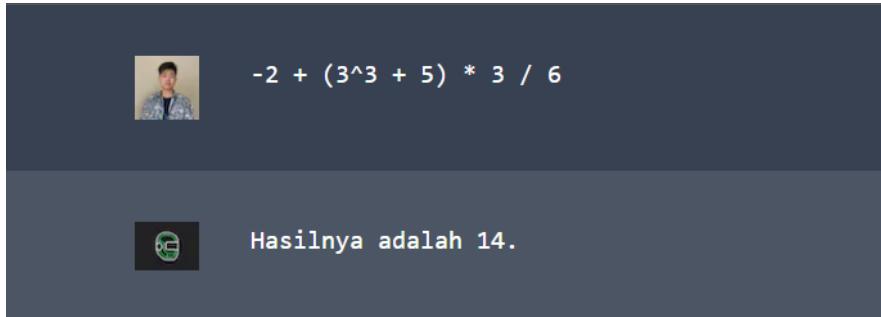
Untuk menggunakan sistem, terdapat beberapa langkah yang perlu diperhatikan, yaitu

1. Pilihlah metode login yang ingin digunakan, apakah dengan akun biasa atau dengan akun google. Jika memilih untuk menggunakan akun biasa, jangan lupa untuk melakukan register terlebih dahulu dengan meng-klik tombol register dan mengisi form register jika belum memiliki akun.
2. Ketika login sudah berhasil, untuk memulai chat, masukkan pesan pertama anda pada textbox di bawah. Halaman chat akan otomatis terbuka untuk suatu sesi/*history*.
3. Untuk memulai chat baru lain, dapat memencet tombol di atas kiri yang bertulisan “+ New Chat”.
4. Untuk mengganti sesi/*history* chat dapat memilih di bagian kiri tengah.
5. Judul/topik dari suatu sesi/*history* dapat diganti dengan memencet tombol pensil dan memasukkan judul/topik baru yang diinginkan lalu memencet tombol centang jika ingin diubah atau silang jika batal.
6. Suatu sesi/*history* chat dapat dihapus dengan memencet tombol hapus dan memencet tombol konfirmasi.
7. Terdapat fitur “Clear conversation” yang dapat menghapus seluruh sesi/*history* yang dimiliki oleh sebuah *user*. Jangan lupa untuk memencet tombol konfirmasi untuk memastikan *user* ingin menghapus seluruh riwayat chatnya.

8. Terdapat menu “Settings” untuk memilih jenis algoritma untuk string matching.
9. Terdapat menu “Get help” untuk memberi petunjuk penggunaan aplikasi.
10. Terdapat opsi “Log out” untuk melakukan proses *logout* dan mengganti akun *user*.
11. Untuk jenis pesan yang di-*handle* oleh aplikasi bisa dibaca pada sub bab 3.2 mengenai fitur fungsional yang terdapat pada aplikasi.

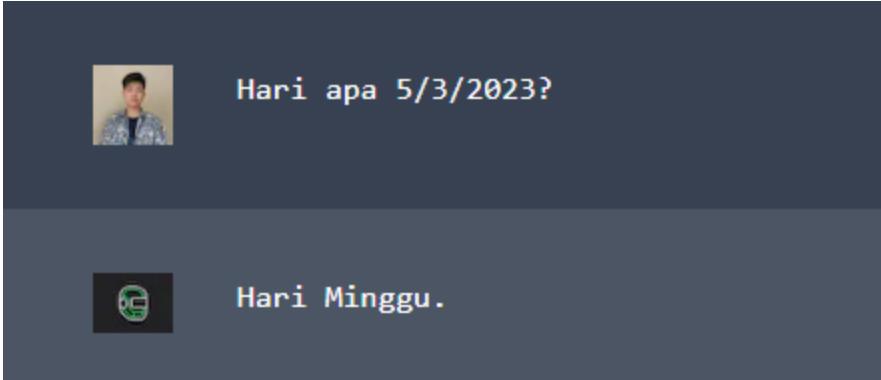
### 4.3. Hasil Pengujian

#### 4.3.1 Pengujian 1

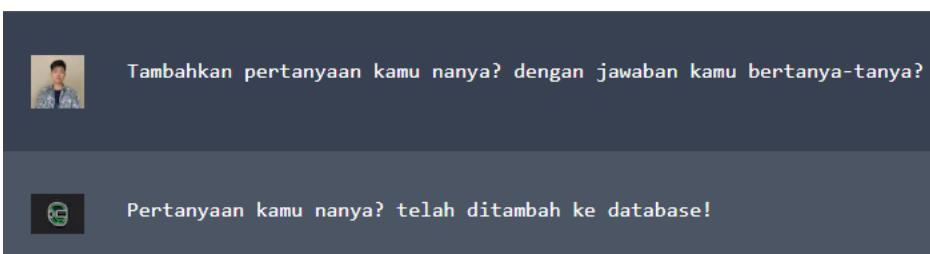
Query	$-2 + (3^3 + 5) * 3 / 6$
Klasifikasi	Kalkulator
Jawaban	Hasilnya adalah 14.
Screenshot	 A screenshot of a mobile application interface. At the top, there is a small circular profile picture of a person. Below it, the mathematical expression $-2 + (3^3 + 5) * 3 / 6$ is displayed. At the bottom, there is a dark bar with a small circular icon containing a white letter 'G' on the left, and the text "Hasilnya adalah 14." on the right.

#### 4.3.2 Pengujian 2

Query	Hari apa 5/3/2023?
Klasifikasi	Tanggal
Jawaban	Hari Minggu.

Screenshot	
------------	--

#### 4.3.3 Pengujian 3

Query	Tambahkan pertanyaan kamu nanya? dengan jawaban kamu bertanya-tanya?
Klasifikasi	Tambah pertanyaan dengan pertanyaan yang belum terdaftar
Jawaban	Pertanyaan kamu nanya? telah ditambah ke database!
Screenshot	

#### 4.3.4 Pengujian 4

Query	Tambahkan pertanyaan kamu nanya? dengan jawaban ya ndak tahu kok nanya saya?
Klasifikasi	Tambah pertanyaan dengan pertanyaan yang sudah terdaftar
Jawaban	Pertanyaan kamu nanya? sudah ada! Jawaban di-update ke ya ndak tahu kok nanya saya?.

Screenshot	 Tambahkan pertanyaan kamu nanya? dengan jawaban ya ndak tahu kok nanya saya?  Pertanyaan kamu nanya? sudah ada! Jawaban di-update ke ya ndak tahu kok nanya saya?.
------------	--

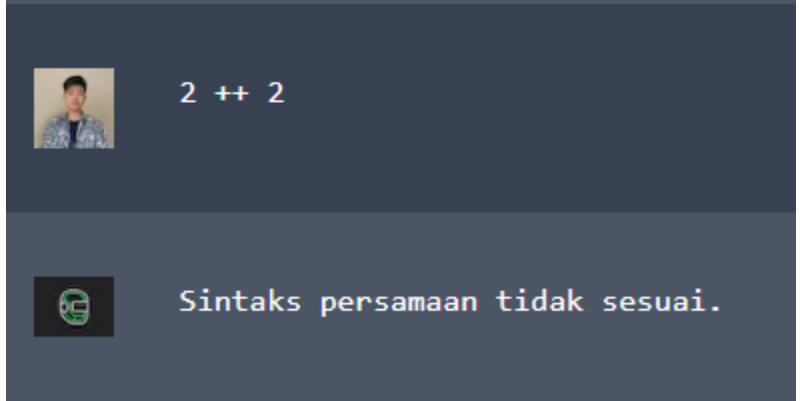
#### 4.3.5 Pengujian 5

Query	Hapus pertanyaan kamu nanya?
Klasifikasi	Hapus pertanyaan yang masih terdapat di database
Jawaban	Pertanyaan kamu nanya? telah dihapus.
Screenshot	 Hapus pertanyaan kamu nanya?  Pertanyaan kamu nanya? telah dihapus.

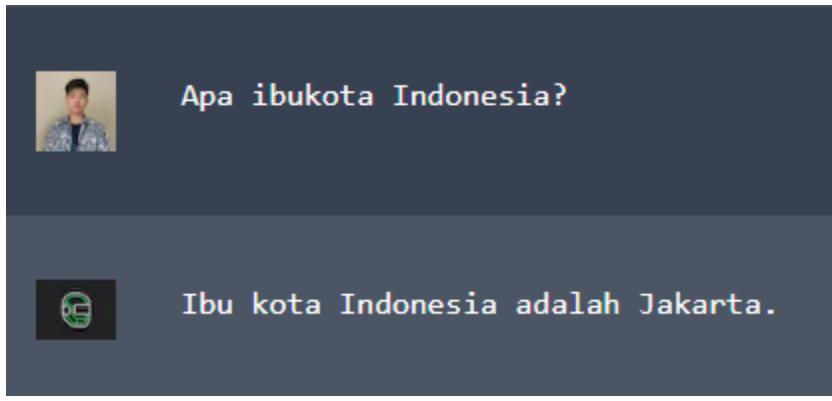
#### 4.3.6 Pengujian 6

Query	Hapus pertanyaan kamu bertanya-tanya?
Klasifikasi	Hapus pertanyaan yang tidak terdapat di database
Jawaban	Tidak ada pertanyaan kamu bertanya-tanya? pada database!
Screenshot	 Hapus pertanyaan kamu bertanya-tanya?  Tidak ada pertanyaan kamu bertanya-tanya? pada database!

#### 4.3.7 Pengujian 7

Query	2 ++ 2
Klasifikasi	Kalkulator dengan sintaks persamaan tidak valid
Jawaban	Sintaks persamaan tidak sesuai.
Screenshot	 A screenshot of a messaging application. At the top, there is a small profile picture of a person. Below it, the text "2 ++ 2" is displayed. In the bottom right corner of the message area, there is a green circular icon with a white question mark. To the right of the icon, the text "Sintaks persamaan tidak sesuai." is written in white.

#### 4.3.8 Pengujian 8

Query	Apa ibukota Indonesia?
Klasifikasi	Pertanyaan yang exact match
Jawaban	Ibu kota Indonesia adalah Jakarta.
Screenshot	 A screenshot of a messaging application. At the top, there is a small profile picture of a person. Below it, the text "Apa ibukota Indonesia?" is displayed. In the bottom right corner of the message area, there is a green circular icon with a white question mark. To the right of the icon, the text "Ibu kota Indonesia adalah Jakarta." is written in white.

#### 4.3.9 Pengujian 9

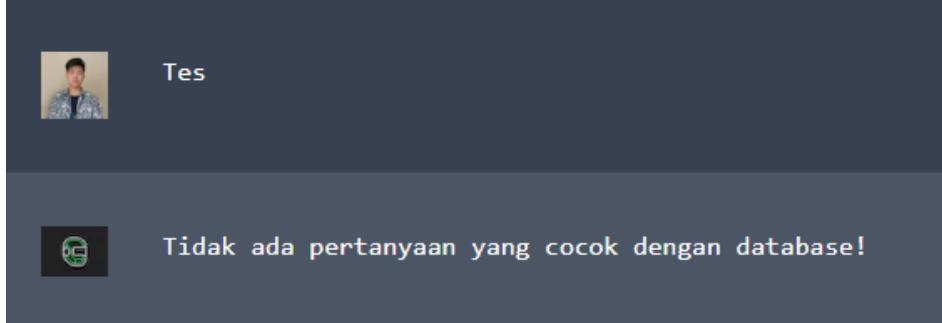
Query	Apa ibukota Indonesia?
-------	------------------------

Klasifikasi	Pertanyaan yang memiliki kemiripan di atas 90%
Jawaban	Ibu kota Indonesia adalah Jakarta.
Screenshot	<p>A screenshot of a messaging interface. The user's message 'Apa ibukota Indonesia?' is shown in blue text. The bot's response 'Ibu kota Indonesia adalah Jakarta.' is shown in white text. There are small profile pictures of the user and the bot next to their respective messages.</p>

#### 4.3.10 Pengujian 10

Query	Apa ibukota?
Klasifikasi	Pertanyaan yang memiliki kemiripan di atas 50%
Jawaban	<p>Pertanyaan tidak ditemukan di database.          Apakah maksud Anda:</p> <ol style="list-style-type: none"> <li>1. Apa itu DNA?</li> <li>2. Apa itu atom?</li> <li>3. Apa ibu kota Indonesia?</li> </ol>
Screenshot	<p>A screenshot of a messaging interface. The user's message 'Apa ibukota?' is shown in blue text. The bot's response is a list of three questions in white text: 'Pertanyaan tidak ditemukan di database.', 'Apakah maksud Anda:', and a numbered list from 1 to 3. There are small profile pictures of the user and the bot next to their respective messages.</p>

#### 4.3.11 Pengujian 11

Query	Tes
Klasifikasi	Pertanyaan yang memiliki kemiripan di bawah 50%
Jawaban	Tidak ada pertanyaan yang cocok dengan database!
Screenshot	

#### 4.4. Analisis Hasil Pengujian

Prioritas pengecekan dilakukan dengan memeriksa apakah query kosong atau tidak, tanggal atau tidak, kalkulator atau tidak, hapus atau tidak, dan delete atau tidak. Jika kelima tersebut tidak cocok juga, maka termasuk menanyakan suatu pertanyaan.

Untuk kasus uji 1, query tersebut diklasifikasikan sebagai fitur kalkulator karena dengan pattern regex yang digunakan, seluruh query yang dikirimkan hanya berisi angka dan operator yang membentuk persamaan matematika. Kemudian, algoritma akan memproses persamaan matematika tersebut dan menentukan apakah sintaks dapat dihitung atau tidak. Dengan algoritma kalkulator menggunakan stack, selama pemrosesan perhitungan juga akan diperiksa apakah sintaks valid atau tidak. Dalam kasus ini sintaks valid sehingga dihasilkan hasil yaitu 14.

Untuk kasus uji 2, query tersebut diklasifikasikan sebagai fitur tanggal karena dengan pattern regex kosong tidak match namun setelah itu dengan pattern regex tanggal berhasil match karena terdapat format tanggal di akhir kalimat. Kemudian, algoritma akan memproses tanggal tersebut untuk menghasilkan hari yang diinginkan.

Untuk kasus uji 3, query tersebut diklasifikasikan sebagai fitur tambah pertanyaan karena dengan pattern regex kosong, tanggal, dan kalkulator tidak match, namun dengan pattern regex tambah pertanyaan berhasil match karena diawali dengan keyword yang tepat yaitu “Tambahkan pertanyaan” diikuti dengan pertanyaan yang ingin ditambahkan, lalu dilanjutkan dengan keyword yang tepat juga yaitu “dengan jawaban” dan diakhiri jawaban yang diinginkan.

Untuk kasus uji 4, query tersebut diklasifikasikan sebagai fitur update pertanyaan karena dengan pattern tambah pertanyaan berhasil namun saat masuk ke dalam algoritma back-end, setelah diiterasi seluruh pertanyaan ditemukan pertanyaan yang exact match dengan pertanyaan yang ingin ditambahkan sehingga jawabannya di-update dengan jawaban terbaru.

Untuk kasus uji 5, query tersebut diklasifikasikan sebagai fitur hapus pertanyaan karena dengan pattern kosong, kalkulator, tanggal, dan tambah pertanyaan tidak match, namun dengan pattern hapus pertanyaan berhasil match sehingga pertanyaan yang exact match dengan query dihapus dari database.

Untuk kasus uji 6, query tersebut diklasifikasikan sebagai fitur hapus pertanyaan juga namun dalam algoritma string matching dengan seluruh pertanyaan di database, tidak ditemukan pertanyaan yang exact match sehingga tidak ditemukan pertanyaan yang ingin dihapus di database.

Untuk kasus uji 7, query tersebut diklasifikasikan sebagai fitur kalkulator juga namun dalam algoritma perhitungan persamaan matematika tidak dapat di-solve sehingga fungsi kalkulator akan melempar error yang membuat bot menjawab sintaks persamaan tidak sesuai.

Untuk kasus uji 8, query tersebut diklasifikasikan sebagai fitur bertanya karena tidak ada satupun pattern regex yang match dengan query tersebut sehingga algoritma

akan melakukan algoritma string matching terhadap seluruh pertanyaan yang di database dan mencari yang exact match dan melemparkannya ke aplikasi.

Untuk kasus uji 9, query tersebut juga diklasifikasikan sebagai fitur bertanya dan pada kasus ini tidak ada yang exact match, namun karena terdapat persentase kemiripannya di atas 90% dengan algoritma string similarity, maka algoritma akan memilih pertanyaan dengan persentase kemiripan tertinggi.

Untuk kasus uji 10, query tersebut juga diklasifikasikan sebagai fitur bertanya namun tidak ada yang exact match maupun persentase kemiripan di atas 90%. Namun, karena terdapat beberapa pertanyaan yang masih memiliki persentase kemiripan di atas 50%, maka akan direkomendasikan maksimal tiga pertanyaan dengan persentase kemiripan tertinggi kepada user.

Untuk kasus uji 11, query tersebut juga diklasifikasikan sebagai fitur bertanya namun karena tidak ada yang exact match dengan query tersebut dan tidak ada pertanyaan yang persentase kemiripannya di atas 50%, maka bot akan menjawab tidak ada pertanyaan yang cocok dengan database.

## **BAB V**

### **PENUTUP**

#### **5.1. Kesimpulan**

Pada Tugas Besar 3 IF2211 Strategi Algoritma ini telah diimplementasikan sebuah aplikasi website chatbot yang dapat menjawab pertanyaan pengguna menggunakan algoritma KMP dan BM juga regex untuk klasifikasi pertanyaan yang telah dipelajari selama akhir semester 4 di mata kuliah Strategi Algoritma dan tambahan materi yang kami eksplor secara mandiri, yaitu algoritma levenshtein distance untuk mencari persentase kemiripan suatu string dengan string lainnya. Program ini dibuat dalam bahasa pemrograman typescript dengan framework Next.js untuk front-end dan Express.js untuk back-end dan dibuat dalam dua repository berbeda untuk front-end dan back-end untuk mempermudah deployment ke web server.

#### **5.2. Saran**

Dari proses penggerjaan tugas besar ini, kami memiliki banyak kendala selama mengerjakan seperti kurangnya informasi dan ambiguitas nama persoalan dari spesifikasi yang telah dibuat. Oleh karena itu, kami memiliki beberapa saran agar tugas ini bisa menjadi lebih baik, yaitu

- a. Pemberian contoh yang lebih jelas dan benar karena ada contoh yang diberikan pada spek yang salah.
- b. Constraint yang lebih jelas untuk implementasi aplikasi.

#### **5.3. Refleksi dan Tanggapan**

Tugas Besar 3 IF2211 Strategi Algoritma ini merupakan salah satu tugas besar yang menantang bagi kami karena banyak sekali hal-hal yang harus dipelajari seperti pembuatan aplikasi yang berbasis website namun seharusnya didapatkan pada semester 5. Namun, dengan tantangan ini, kami mendapatkan kesempatan untuk melakukan eksplorasi lebih dalam sehingga tugas besar ini sangat mengembangkan skill yang kami miliki. Tantangan yang dialami adalah bagaimana aplikasi yang kami buat bisa memiliki

fungisionalitas yang mirip dengan aplikasi ChatGPT. Tantangan ini dihadapi dengan mempelajari lebih lanjut pengimplementasian dan library-library dari framework.

Tanggapan dari kelompok kami mengenai tugas ini adalah tugas ini merupakan salah satu tugas yang cukup seru untuk dibahas dan dapat merasakan kepuasan setelah menyelesaikan aplikasi dengan baik karena aplikasi dapat dibuat dengan cantik dan efisien. Masa-masa eksplorasi juga kadang membuat sakit kepala namun menjadi sebuah kepuasan bagi kami ketika suatu permasalahan dapat diperbaiki dan diatasi dengan baik.

## **DAFTAR PUSTAKA**

- Munir, Rinaldi. 2023. Pencocokan string (String matching/ pattern matching).. Homepage Rinaldi Munir Sekolah Teknik Elektro dan Informatika (STEI) ITB.  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf>
- Munir, Rinaldi. 2023. Pencocokan string dengan Regular Expression (Regex). Homepage Rinaldi Munir Sekolah Teknik Elektro dan Informatika (STEI) ITB.  
<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf>

## **LAMPIRAN**

- |                       |   |   |
|-----------------------|---|---|
| Link Github Front-End | : | <a href="https://github.com/maikeljh/Tubes3_Athena_FE">https://github.com/maikeljh/Tubes3_Athena_FE</a>                             |
| Link Github Back-End  | : | <a href="https://github.com/maikeljh/Tubes3_Athena_BE">https://github.com/maikeljh/Tubes3_Athena_BE</a>                             |
| Link Deployment       | : | <a href="https://athena-stima.vercel.app/">https://athena-stima.vercel.app/</a>   |
| Link Video            | : | <a href="https://youtu.be/Hj-8GXPWuTU">https://youtu.be/Hj-8GXPWuTU</a>   |
| Link Dokumentasi API  | : | <a href="https://documenter.getpostman.com/view/17084544/2s93eVXtc9">https://documenter.getpostman.com/view/17084544/2s93eVXtc9</a> |