

**Laporan Tugas Besar 3**  
**IF2211 Strategi Algoritma**

**Penerapan *String Matching* dan *Regular Expression*  
dalam *DNA Pattern Matching***



oleh :

Kelompok 48 testDNA

13520135    Muhammad Alif Putra Yasa

13520158    Azmi Alfatih Shalahuddin

13520165    Ghazian Tsabit Alkamil

**Sekolah Teknik Elektro dan Informatika**  
**Program Studi Teknik Informatika**  
**Institut Teknologi Bandung**  
**Tahun Ajaran 2021/2022**

## **DAFTAR ISI**

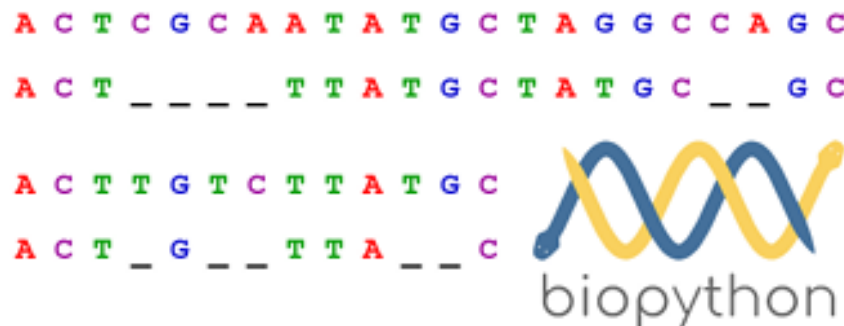
<b>BAB I</b>	<b>3</b>
<b>Deskripsi Tugas</b>	<b>3</b>
Latar Belakang	3
Deskripsi Tugas	4
Fitur-fitur Aplikasi	4
D. Spesifikasi Program	9
<b>BAB II</b>	<b>10</b>
<b>Landasan Teori</b>	<b>10</b>
Algoritma Knuth-Morris-Pratt	10
Algoritma Boyer-Moore	10
Regular Expression	11
DNA Pattern Matching Web App	12
<b>BAB III</b>	<b>13</b>
<b>Analisis Pemecahan Masalah</b>	<b>13</b>
Langkah Penyelesaian Fitur Tambah Jenis Penyakit	13
Langkah Penyelesaian Fitur Prediksi Penyakit	16
Langkah Penyelesaian Fitur Pencarian Hasil Prediksi	20
Fitur Aplikasi dan Arsitektur Web Aplikasi	22
Fitur Fungsional	22
Arsitektur Web Aplikasi	22
<b>BAB IV</b>	<b>24</b>
<b>Implementasi dan Pengujian</b>	<b>24</b>
Spesifikasi Teknis Program	24
Tata Cara Penggunaan Program	26
Hasil Pengujian	27
Analisis Hasil Pengujian	29
<b>BAB V</b>	<b>31</b>
<b>Kesimpulan dan Saran</b>	<b>31</b>
<b>DAFTAR PUSTAKA</b>	<b>32</b>

## BAB I

### Deskripsi Tugas

#### A. Latar Belakang

Manusia umumnya memiliki 46 kromosom di dalam setiap selnya. Kromosom-kromosom tersebut tersusun dari DNA (deoxyribonucleic acid) atau asam deoksiribonukleat. DNA tersusun atas dua zat basa purin, yaitu Adenin (A) dan Guanin (G), serta dua zat basa pirimidin, yaitu sitosin (C) dan timin (T). Masing-masing purin akan berikatan dengan satu pirimidin. DNA merupakan materi genetik yang menentukan sifat dan karakteristik seseorang, seperti warna kulit, mata, rambut, dan bentuk wajah. Ketika seseorang memiliki kelainan genetik atau DNA, misalnya karena penyakit keturunan atau karena faktor lainnya, ia bisa mengalami penyakit tertentu. Oleh karena itu, tes DNA penting untuk dilakukan untuk mengetahui struktur genetik di dalam tubuh seseorang serta mendeteksi kelainan genetik. Ada berbagai jenis tes DNA yang dapat dilakukan, seperti uji pra implantasi, uji pra kelahiran, uji pembawa atau carrier testing, uji forensik, dan DNA sequence analysis.



Gambar 1. Ilustrasi Sekuens DNA

Sumber :

<https://towardsdatascience.com/pairwise-sequence-alignment-using-biopython-d1a9d0ba861f>

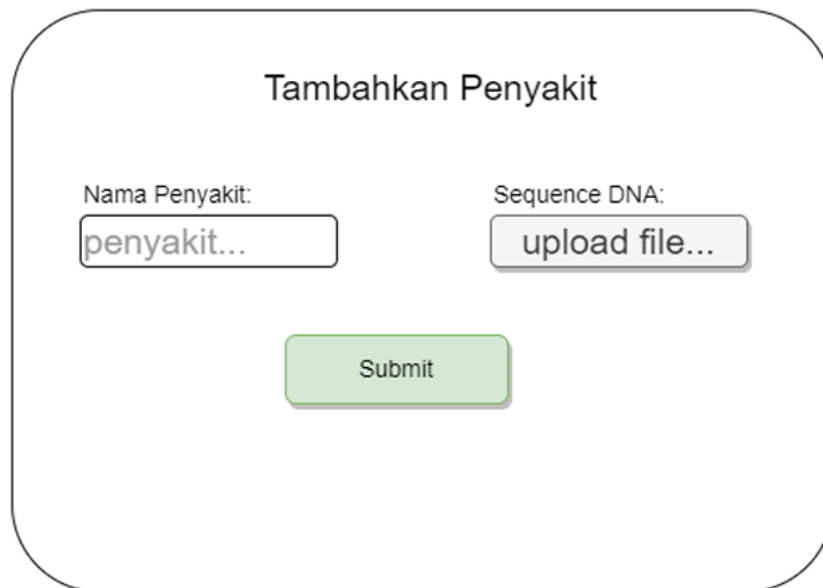
Salah satu jenis tes DNA yang sangat berkaitan dengan dunia bioinformatika adalah DNA sequence analysis. DNA sequence analysis adalah sebuah cara yang dapat digunakan untuk memprediksi berbagai macam penyakit yang tersimpan pada database berdasarkan urutan sekuens DNA-nya. Sebuah sekuens DNA adalah suatu representasi string of nucleotides yang disimpan pada suatu rantai DNA, sebagai contoh: ATTCGTAAGTAAAGTTA. Teknik pattern matching memegang peranan penting untuk dapat menganalisis sekuens DNA yang sangat panjang dalam waktu singkat. Oleh karena itu, mahasiswa Teknik Informatika berniat untuk membuat suatu aplikasi web berupa DNA Sequence Matching yang menerapkan algoritma String Matching dan Regular Expression untuk membantu penyedia jasa kesehatan dalam memprediksi penyakit pasien. Hasil prediksi juga dapat ditampilkan dalam tabel dan dilengkapi dengan kolom pencarian untuk membantu admin dalam melakukan filtering dan pencarian.

## **B. Deskripsi Tugas**

Dalam tugas besar ini, anda diminta untuk membangun sebuah aplikasi DNA Pattern Matching. Dengan memanfaatkan algoritma String Matching dan Regular Expression yang telah anda pelajari di kelas IF2211 Strategi Algoritma, anda diharapkan dapat membangun sebuah aplikasi interaktif untuk mendeteksi apakah seorang pasien mempunyai penyakit genetik tertentu. Hasil prediksi tersebut dapat disimpan pada basis data untuk kemudian dapat ditampilkan berdasarkan query pencarian.

## **C. Fitur-fitur Aplikasi**

1. Aplikasi dapat menerima input penyakit baru berupa nama penyakit dan sequence DNA-nya (dan dimasukkan ke dalam database).
  - a. Implementasi input sequence DNA dalam bentuk file.
  - b. Dilakukan sanitasi input menggunakan **regex** untuk memastikan bahwa masukan merupakan sequence DNA yang valid (tidak boleh ada huruf kecil, tidak boleh ada huruf selain AGCT, dan tidak ada spasi).
  - c. Contoh input penyakit:



The image shows a web form titled "Tambahkan Penyakit" (Add Disease). It contains two input fields: "Nama Penyakit:" (Disease Name) with a placeholder text "penyakit..." and "Sequence DNA:" with a placeholder text "upload file...". Below these fields is a green "Submit" button.

Gambar 2. Ilustrasi Input Penyakit

2. Aplikasi dapat memprediksi seseorang menderita penyakit tertentu berdasarkan sequence DNA-nya.
  - a. Tes DNA dilakukan dengan menerima input nama pengguna, sequence DNA pengguna, dan nama penyakit yang diuji. Asumsi sequence DNA pengguna > sequence DNA penyakit.
  - b. Dilakukan sanitasi *input* menggunakan **regex** untuk memastikan bahwa masukan merupakan sequence DNA yang valid (tidak boleh ada huruf kecil, tidak boleh ada huruf selain AGCT, tidak ada spasi, dll).
  - c. Pencocokan sequence DNA dilakukan dengan menggunakan algoritma **string matching**.
  - d. Hasil dari tes DNA berupa tanggal tes, nama pengguna, nama penyakit yang diuji, dan status hasil tes. **Contoh: 1 April 2022- Mhs IF - HIV - False**
  - e. Semua komponen hasil tes ini dapat ditampilkan pada halaman web (refer ke poin 3 pada “Fitur-Fitur Aplikasi”) dan disimpan pada sebuah tabel database.
  - f. Contoh tampilan web:

**Tes DNA**

Nama Pengguna:

Sequence DNA:

Prediksi Penyakit:

---

**Hasil Tes**

**<Tanggal> - <pengguna> - <penyakit> - <True/False>**

Gambar 3. Ilustrasi Prediksi

3. Aplikasi memiliki halaman yang menampilkan urutan hasil prediksi dengan kolom pencarian di dalamnya. Kolom pencarian bekerja sebagai filter dalam menampilkan hasil.
  - a. Kolom pencarian dapat menerima masukan dengan struktur: <tanggal\_prediksi><spasi><nama\_penyakit>, contoh “13 April 2022 HIV”. **Format penanggalan dibebaskan**, jika bisa menerima >1 format lebih baik.
  - b. Kolom pencarian dapat menerima masukan hanya tanggal ataupun hanya nama penyakit. Fitur ini diimplementasikan menggunakan **regex**.

- c. Contoh ilustrasi:
- Masukan tanggal dan nama penyakit

13 April 2022 HIV

1. 13 April 2022 - Fulan - HIV - True.

2. 13 April 2022 - Karmel - HIV - False.

3. 13 April 2022 - Entah - HIV - False.

4. 13 April 2022 - Jamal - HIV - True.

5. 13 April 2022 - Yubel - HIV - True.

6. 13 April 2022 - Hika - HIV - False.

Gambar 4. Ilustrasi Interaksi 1

- Masukan hanya tanggal

The image shows a web application interface. At the top, there is a date input field containing "13 April 2022". Below this, there is a list of six entries, each in a separate box. The entries are numbered 1 through 6. Each entry contains a date, a name, and a medical condition with a boolean value.

No	Date	Name	Condition	Value
1.	13 April 2022	Fulan	Diabetes	True
2.	13 April 2022	Kamal	Sinusitis	False
3.	13 April 2022	Entah	Down Syndrome	False
4.	13 April 2022	Jamal	Polio	True
5.	13 April 2022	Yubai	TBC	True
6.	13 April 2022	Hika	Hepatitis A	False

Gambar 5. Ilustrasi Interaksi 2

iii. Masukkan hanya nama penyakit

HIV

1. 13 April 2022 - Fulan - HIV - True.

2. 14 April 2022 - Kamel - HIV - False.

3. 15 April 2022 - Entah - HIV - False.

4. 16 April 2022 - Jamal - HIV - True.

5. 17 April 2022 - Yubai - HIV - True.

6. 18 April 2022 - Hika - HIV - False.

Gambar 6. Ilustrasi Interaksi 3

4. (Bonus) Menghitung tingkat kemiripan DNA pengguna dengan DNA penyakit pada tes DNA
  - a. Ketika melakukan tes DNA, terdapat persentase kemiripan DNA dalam hasil tes.  
**Contoh hasil tes: 1 April 2022 - Mhs IF - HIV - 75% - False**
  - b. Perhitungan tingkat kemiripan dapat dilakukan dengan menggunakan Hamming distance, Levenshtein distance, LCS, atau algoritma lainnya (dapat dijelaskan dalam laporan).
  - c. Tingkat kemiripan DNA dengan nilai lebih dari atau sama dengan 80% dikategorikan sebagai **True**. Perlu diperhatikan mengimplementasikan atau tidak mengimplementasikan bonus ini tetap dilakukan pengecekan string matching terlebih dahulu.
  - d. Contoh tampilan:



**Tes DNA**

Nama Pengguna:

Sequence DNA:

Prediksi Penyakit:

---

**Hasil Tes**

<Tanggal> - <pengguna> - <penyakit> - <similarity> - <True/False>

Gambar 7. Output Tingkat Kemiripan

#### D. Spesifikasi Program

1. Aplikasi berbasis website dengan pembagian Frontend dan Backend yang jelas.
2. Implementasi Backend **wajib** menggunakan Node.js / Golang, sedangkan Frontend **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) **wajib** diimplementasikan pada sisi Backend aplikasi.
5. Informasi yang **wajib** disimpan pada basis data:
  - a. Jenis Penyakit:
    - Nama Penyakit
    - Rantai DNA penyusun
  - b. Hasil Prediksi:
    - Tanggal prediksi
    - Nama pasien
    - Penyakit prediksi
    - Status prediksi
6. Jika mengerjakan bonus tingkat kemiripan DNA, simpan hasil tingkat kemiripan tersebut pada basis data.

## **BAB II**

### **Landasan Teori**

#### **A. Algoritma Knuth-Morris-Pratt**

Algoritma The Knuth-Morris-Pratt (KMP) adalah sebuah algoritma pattern matching yang mencari pattern dalam text dalam urutan kiri ke kanan seperti algoritma Brute Force. Akan tetapi, algoritma ini melakukan pattern shift lebih baik dibanding algoritma brute force.

##### **Kompleksitas Waktu KMP**

1. Menghitung fungsi pinggiran :  $O(m)$ ,
2. Pencarian string :  $O(n)$
3. Kompleksitas waktu algoritma KMP adalah  $O(m+n)$ . (sangat cepat dibanding brute force)

##### **Keuntungan KMP**

1. Algoritma tidak perlu mundur dari text input.
2. Algoritma baik untuk memproses file yang sangat besar.

##### **Kerugian KMP**

1. KMP tidak baik untuk variasi character yang banyak
2. Variasi character yang banyak menyebabkan peluang pattern sesuai semakin sedikit

#### **B. Algoritma Boyer-Moore**

Algoritma Boyer-Moore berdasarkan dua jenis teknik, yaitu looking-glass technique dan character-jump technique.

Terdapat tiga kasus yang dapat terjadi:

1. Case 1  
Jika P mengandung x, coba geser P ke kanan untuk menyelaraskan kemunculan terakhir x di P dengan  $T[i]$ .
2. Case 2  
Jika P berisi x di suatu tempat, tetapi bergeser ke kanan ke yang terakhir kemunculannya tidak mungkin, lalu geser P ke kanan sebanyak 1 karakter ke  $T[i+1]$ .
3. Case 3  
Jika kasus 1 dan 2 tidak berlaku, maka geser P untuk meratakan  $P[0]$  dengan  $T[i+1]$ .

Analisis algoritma boyer-moore:

1. Boyer-Moore worst case memiliki kompleksitas waktu  $O(nm + A)$
2. Boyer-Moore baik untuk variasi karakter banyak (contoh: bahasa inggris) namun buruk untuk variasi sedikit (contoh: binary).

- Boyer-Moore jauh lebih cepat dari brute force untuk searching bahasa inggris.

### C. Regular Expression

Regex adalah singkatan dari regular expression. Regex adalah kumpulan karakter yang menjelaskan pola suatu text. Biasanya, regex digunakan untuk melakukan pencarian pada string atau untuk validasi input.

Beberapa pola regex yang umum digunakan:

- Bracket ([ ])

#### **brackets [ ] : disjunction**

RE	Match	Example Patterns
/ [wW] oodchuck /	Woodchuck or woodchuck	" <u>W</u> oodchuck"
/ [abc] /	'a', 'b', or 'c'	"In uomini, in soldat <u>i</u> "
/ [1234567890] /	any digit	"plenty of <u>7</u> to 5"

#### **Brackets [ ] ditambah garis sambung: range**

RE	Match	Example Patterns Matched
/ [A-Z] /	an uppercase letter	"we should call it ' <u>D</u> renched Blossoms'"
/ [a-z] /	a lowercase letter	" <u>m</u> y beans were impatient to be hoed!"
/ [0-9] /	a single digit	"Chapter <u>1</u> : Down the Rabbit Hole"

- Caret (^)

RE	Match (single characters)	Example Patterns Matched
[^A-Z]	not an uppercase letter	"O <u>y</u> fn pripetchik"
[^Ss]	neither 'S' nor 's'	" <u>I</u> have no exquisite reason for't"
[^\.]	not a period	" <u>o</u> ur resident Djinn"
[e^]	either 'e' or '^'	"look up <u>^</u> now"
a^b	the pattern 'a^b'	"look up <u>a^</u> b now"

- Tanda tanya (?)

RE	Match	Example Patterns Matched
woodchucks?	woodchuck or woodchucks	" <u>w</u> oodchuck"
colou?r	color or colour	" <u>c</u> olour"

#### D. Titik(.)

RE	Match	Example Patterns
/beg.n/	any character between <i>beg</i> and <i>n</i>	<u>begin</u> , <u>beg'n</u> , <u>begun</u>

8

#### D. DNA Pattern Matching Web App

Web aplikasi DNA *pattern matching* adalah web aplikasi yang berguna untuk memprediksi penyakit yang diderita oleh seseorang dengan memproses rantai DNA dari orang tersebut. Web aplikasi ini memanfaatkan algoritma *pattern matching knuth-morris-pratt* untuk menentukan tingkat kesamaan rantai DNA yang dimiliki oleh seseorang dengan rantai DNA suatu penyakit yang terdapat di basis data, tetapi algoritma *pattern matching Boyer-Moore* juga berhasil di implementasikan meskipun pada akhirnya tidak digunakan untuk memprediksi penyakit pengguna.

Apabila tingkat kesamaan nya lebih besar atau sama dengan 80% maka dapat dikatakan orang tersebut menderita penyakit tertentu. Web aplikasi ini juga memanfaatkan *regular expression* untuk melakukan sanitasi terhadap rantai DNA yang di input oleh pengguna, rantai DNA yang valid adalah rantai DNA yang hanya terdiri dari huruf ACGT dan semuanya harus kapital dan juga tanpa spasi.

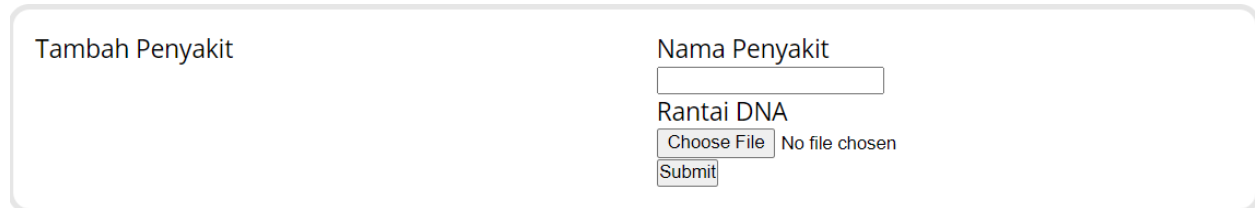
Web aplikasi ini memiliki beberapa fitur sebagai berikut:

1. Menambahkan jenis penyakit
2. Memprediksi penyakit pengguna
3. Mencari hasil prediksi penyakit berdasarkan tanggal dilakukannya prediksi
4. Mencari hasil prediksi penyakit berdasarkan nama penyakit
5. Mencari hasil prediksi penyakit berdasarkan tanggal dan nama penyakit

## BAB III

### Analisis Pemecahan Masalah

#### A. Langkah Penyelesaian Fitur Tambah Jenis Penyakit



Gambar 3.1 Tampilan Fitur Tambah Jenis Penyakit

Fitur tambah jenis penyakit diselesaikan dengan beberapa langkah baik dari sisi *frontend* maupun dari sisi *backend*. Lingkungan implementasi web aplikasi DNA *pattern matching* menggunakan bahasa pemrograman Go pada sisi *backend*, menggunakan framework *React JS* pada sisi *frontend*, dan menggunakan DBMS Postgresql untuk sisi basis data nya.

Pertama dari sisi *backend* dibuat sebuah `struct` yang diberi nama `JenisPenyakit`, `struct JenisPenyakit` memiliki atribut `ID` yang bertipe `int64`, `Nama` yang bertipe `string`, dan `RantaiDNA` yang bertipe `string`.

```
type JenisPenyakit struct {  
    ID          int64  `json:"id"`  
    Nama        string `json:"nama"`  
    RantaiDNA   string `json:"rantai_dna"`  
}
```

Gambar 3.2 Struct JenisPenyakit

Selain itu, dibuat juga sebuah tabel pada database yang bernama `jenis_penyakit` yang memiliki detail yang dapat dilihat pada gambar 3.3.

Table "public.jenis_penyakit"				
Column	Type	Collation	Nullable	Default
id	integer		not null	nextval('jenis_penyakit_id_seq'::regclass)
nama	character varying		not null	
rantai_dna	character varying		not null	
Indexes:				
"jenis_penyakit_pkey" PRIMARY KEY, btree (id)				

Gambar 3.3 Tabel jenis\_penyakit

Setelah *struct JenisPenyakit* dan tabel *jenis\_penyakit* telah berhasil dibuat, langkah selanjutnya adalah dengan membuat beberapa fungsi untuk menambahkan data dengan tipe *JenisPenyakit* ke dalam tabel *jenis\_penyakit* pada basis data.

```
func InsertJenisPenyakit(db *sql.DB, rDNA JenisPenyakit) error {  
    var err1 error  
    sqlQuery := `INSERT INTO jenis_penyakit (nama, rantai_dna) VALUES($1, $2) RETURNING id;`  
    id := 0  
    dnaSanitized, _ := smalgorithm.IsValid(rDNA.RantaiDNA)  
  
    if dnaSanitized {  
        err := db.QueryRow(sqlQuery, rDNA.Nama, rDNA.RantaiDNA).Scan(&id)  
        if err != nil {  
            return fmt.Errorf("SOMEHOW ERROR: %v", err)  
        }  
    } else {  
        err1 = errors.New("rantai dna tidak valid")  
    }  
  
    return err1  
}
```

Gambar 3.4 Fungsi InsertJenisPenyakit

Gambar 3.4 menunjukkan fungsi *InsertJenisPenyakit*, fungsi ini akan digunakan untuk menambahkan sebuah data bertipe *JenisPenyakit* ke dalam tabel *jenis\_penyakit*. Namun, sebelum data yang baru dimasukkan ke dalam basis data, perlu diperiksa terlebih dahulu apakah *RantaiDNA* merupakan rantai DNA yang valid. Rantai DNA yang valid adalah rantai DNA yang tidak ada huruf kecil, tidak boleh ada huruf selain ACGT, dan tidak ada spasi. Rantai DNA diperiksa apakah valid atau tidak dengan menggunakan fungsi *isValid* yang memanfaatkan *regular expression*.

```
func IsValid(s string) (bool, error) {  
    var dnaSanitized bool  
    var err error  
  
    // Gak menerima string kosong  
    if s == "" {  
        return false, nil  
    }  
  
    // Apakah ada yang bukan ACGT  
    dnaSanitized, err = regexp.MatchString("[^ACGT]", s)  
    return (!dnaSanitized), err  
}
```

Gambar 3.5 Fungsi IsValid

Pada fitur tambah penyakit juga perlu dibuat sebuah REST API yang berfungsi sebagai media perantara atau media komunikasi antara sisi *frontend* dengan sisi *backend*. Pembuatan REST API pada web aplikasi ini menggunakan *library gin* pada bahasa pemrograman Go. Fungsi post

request pada fitur tambah penyakit dapat dilihat pada gambar 3.6. Apabila input rantai DNA tidak valid maka fungsi akan menghasilkan *response bad request*, sehingga nantinya bagian *frontend* dapat mengetahui pesan kesalahan yang dikirimkan dari sisi *backend*.

```
func postJenisPenyakit(c *gin.Context) {
    var newJenisPenyakit dbhandler.JenisPenyakit
    if err := c.BindJSON(&newJenisPenyakit); err != nil {
        return
    }
    err1 := dbhandler.InsertJenisPenyakit(db, newJenisPenyakit)
    if err1 != nil {
        fmt.Println("Rantai DNA yang di input tidak valid")
        c.JSON(http.StatusBadRequest, gin.H{
            "status": "KO",
            "message": "rantai dna tidak valid",
            "data": err1,
        })
        return
    } else {
        c.IndentedJSON(http.StatusCreated, newJenisPenyakit)
    }
}
```

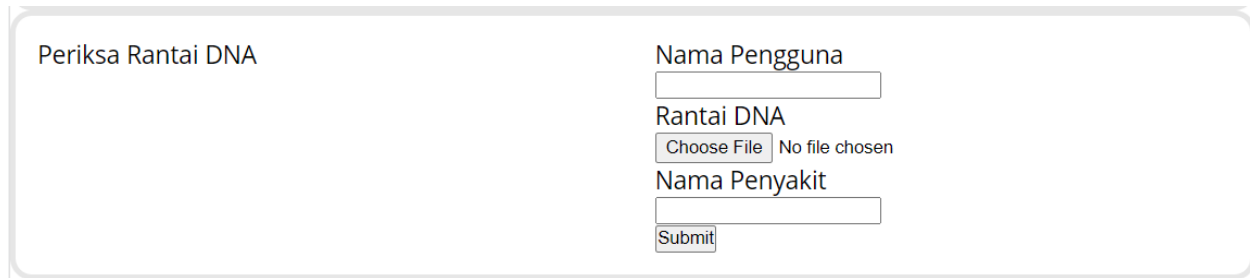
Gambar 3.6 Fungsi postJenisPenyakit

Dari sisi *frontend* telah dibuat sebuah fungsi *handlesubmit*, yang akan melakukan *post request* kepada sisi *backend* ketika tombol *submit* yang tersedia ditekan. Data yang di *post* adalah data yang sebelumnya telah diisikan oleh pengguna pada *form* yang tersedia.

```
axios({
    method: 'POST',
    url: Endpoints.addPenyakit,
    headers: {
        'Content-Type': 'application/json'
    },
    data: {
        id: this.id,
        nama: this.nama_penyakit.value,
        rantai_dna: this.state.rantai_dna
    }
}).then((response) => {
    console.log(response);
}, (error) => {
    console.log(error.message);
    this.setState({
        status: "Rantai DNA tidak valid"
    })
});
```

3.7 Fungsi Post Request Frontend

## B. Langkah Penyelesaian Fitur Prediksi Penyakit



The screenshot shows a web interface for disease prediction. On the left, there is a label 'Periksa Rantai DNA'. On the right, there are several input fields and a submit button. The fields are labeled 'Nama Pengguna', 'Rantai DNA', and 'Nama Penyakit'. The 'Rantai DNA' field has a 'Choose File' button and the text 'No file chosen' next to it. Below the 'Nama Penyakit' field is a 'Submit' button.

Gambar 3.8 Tampilan Fitur Prediksi Penyakit

Fitur prediksi penyakit diselesaikan dengan beberapa langkah baik dari sisi *frontend* maupun dari sisi *backend*. Lingkungan implementasi web aplikasi *DNA pattern matching* menggunakan bahasa pemrograman *Go* pada sisi *backend*, menggunakan framework *React JS* pada sisi *frontend*, dan menggunakan DBMS *Postgresql* untuk sisi basis data nya.

Pertama dari sisi *backend* dibuat sebuah struct yang diberi nama *Pasien*, struct *Pasien* memiliki atribut *IdPengguna* yang bertipe *int64*, *NamaPengguna* yang bertipe *string*, *RantaiDNA* yang bertipe *string*, *NamaPenyakit* yang bertipe *string*, dan *TanggalPrediksi* yang bertipe *string*.

```
type Pasien struct {  
    IdPengguna      int64  `json:"id"`  
    NamaPengguna    string `json:"nama_pengguna"`  
    RantaiDNA       string `json:"rantai_dna"`  
    NamaPenyakit    string `json:"nama_penyakit"`  
    TanggalPrediksi string `json:"tanggal_prediksi"`  
}
```

Gambar 3.8 struct Pasien

Selain struct *Pasien*, perlu juga dibuat struct *HasilPrediksi*. Struct *HasilPrediksi* memiliki beberapa atribut yaitu *ID* yang bertipe *int64*, *TanggalPrediksi* yang bertipe *string*, *NamaPasien* yang bertipe *string*, *NamaPenyakit* yang bertipe *string*, *TingkatKemiripan* yang bertipe *float64*, dan atribut *StatusPrediksi* yang bertipe *bool*.



```
type HasilPrediksi struct {
    ID                int64    `json:"id"`
    TanggalPrediksi string    `json:"tanggal_prediksi"`
    NamaPasien        string    `json:"nama_pasien"`
    NamaPenyakit       string    `json:"nama_penyakit"`
    TingkatKemiripan  float64  `json:"tingkat_kemiripan"`
    StatusPrediksi   bool     `json:"status_prediksi"`
}
```

Gambar 3.9 struct HasilPrediksi

Dibuat juga sebuah tabel pada database yang bernama `hasil_prediksi` yang memiliki detail yang dapat dilihat pada gambar 3.10.

Table "public.hasil_prediksi"				
Column	Type	Collation	Nullable	Default
id	integer		not null	nextval('hasil_prediksi_id_seq'::regclass)
tanggal_prediksi	date		not null	CURRENT_DATE
nama_pasien	character varying		not null	
nama_penyakit	character varying		not null	
tingkat_kemiripan	numeric(3,2)		not null	
status_prediksi	boolean		not null	

Indexes:

"hasil\_prediksi\_pkey" PRIMARY KEY, btree (id)

Gambar 3.10 Tabel hasil\_prediksi

Pada fitur prediksi penyakit juga perlu dibuat sebuah REST API yang berfungsi sebagai media komunikasi antara sisi *frontend* dengan sisi *backend*. Pembuatan REST API pada web aplikasi ini menggunakan *library gin* pada bahasa pemrograman *Go*. Fungsi post request pada fitur prediksi penyakit dapat dilihat pada gambar 3.11.

Skema dari fungsi *postPasien* adalah sebagai berikut:

1. Program menerima sebuah data baru bertipe Pasien
2. Diperiksa apakah atribut RantaiDNA pada data Pasien tersebut valid atau tidak menggunakan fungsi *isValid* yang memanfaatkan *regular expression*.
3. Apabila RantaiDNA tidak valid maka fungsi akan mengirimkan *response* dengan status *bad request*, apabila RantaiDNA valid maka fungsi *postPasien* akan memanggil fungsi *periksaPenyakit* untuk memeriksa apakah RantaiDNA yang diinput sesuai dengan RantaiDNA penyakit yang ingin diperiksa.
4. Apabila pada saat pemanggilan fungsi *periksaPenyakit* terdapat error data penyakit tidak ditemukan di basis data, maka program akan mengembalikan *response status no content*, sehingga sisi *frontend* dapat menangkap pesan tersebut.

5. Apabila pada saat pemanggilan fungsi *periksaPenyakit* tidak terdapat error maka data *Pasien* yang baru berhasil ditambahkan dan tabel *hasil\_prediksi* pada basis data akan bertambah datanya sesuai keadaan RantaiDNA pada saat itu.

```
func postPasien(c *gin.Context) {
    var newPasien Pasien
    if err := c.BindJSON(&newPasien); err != nil {
        return
    }
    dnaSanitized, _ := smalgorithm.IsValid(newPasien.RantaiDNA)
    if dnaSanitized {
        pasien = append(pasien, newPasien)
        err := newPasien.periksaPenyakit()
        if err != nil {
            c.JSON(http.StatusNoContent, gin.H{
                "status": "KO",
                "message": "Data penyakit tidak ditemukan",
                "data":    dnaSanitized,
            })
        } else {
            c.IndentedJSON(http.StatusCreated, newPasien)
        }
    } else {
        // fmt.Println("Rantai DNA yang di input tidak valid")
        c.JSON(http.StatusBadRequest, gin.H{
            "status": "KO",
            "message": "rantai dna tidak valid",
            "data":    dnaSanitized,
        })
    }
}
```

Gambar 3.11 Fungsi postPasien

Fungsi dari *postPasien* memanggil fungsi lain yaitu fungsi *periksaPenyakit*, skema dari fungsi *periksaPenyakit* adalah sebagai berikut :

1. Fungsi akan mengambil data RantaiDNA pada basis data sesuai dengan nama penyakit yang ada pada data Pasien. Apabila terdapat error pada saat pengambilan data tersebut maka program akan mengembalikan error tersebut.
2. Apabila pada saat pengambilan data tidak terdapat error maka langkah selanjutnya adalah diperiksa lagi apakah RantaiDNA yang terdapat pada data Pasien merupakan rantai DNA yang valid.
3. Apabila RantaiDNA valid, maka langkah selanjutnya adalah program akan memanggil fungsi *KMPMod* yang akan memeriksa tingkat kemiripan dari RantaiDNA yang terdapat pada Pasien dengan RantaiDNA yang terdapat pada basis data.

4. Apabila tingkat kemiripan lebih dari atau sama dengan 0.8 maka status kemiripan rantai DNA tersebut bernilai *true*, sedangkan apabila tingkat kemiripan kurang dari 0.8 maka status kemiripan rantai DNA tersebut bernilai *false*.
5. Setelah nilai dari tingkat kemiripan dan status kemiripan didapatkan maka langkah selanjutnya adalah menambahkan data baru pada tabel *hasil\_prediksi*.
6. Detail dari fungsi *periksaPenyakit* dapat dilihat pada gambar 3.12

```
func (p Pasien) periksaPenyakit() error {  
    // kamus lokal  
    var rantai_dna string  
    var mirip bool  
    var err error  
    sqlQuery := `SELECT rantai_dna FROM jenis_penyakit WHERE nama = $1;`  
    err1 := db.QueryRow(sqlQuery, p>NamaPenyakit).Scan(&rantai_dna)  
    if err1 != nil {  
        err = err1  
    } else {  
        fmt.Println("\n", rantai_dna)  
        dnaSanitized, err := smalgorithm.IsValid(p.RantaiDNA)  
        if dnaSanitized {  
            _, diff, _ := smalgorithm.KMPMod(p.RantaiDNA, rantai_dna)  
            if diff >= 0.8 {  
                mirip = true  
            } else {  
                mirip = false  
            }  
            sqlQuery := `INSERT INTO hasil_prediksi (tanggal_prediksi, nama_pasien,  
                nama_penyakit, tingkat_kemiripan, status_prediksi) VALUES($1, $2, $3, $4, $5) RETURNING id;`  
            id := 0  
            err = db.QueryRow(sqlQuery, p.TanggalPrediksi, p>NamaPengguna, p>NamaPenyakit, diff, mirip).Scan(&id)  
            if err != nil {  
                panic(err)  
            }  
        } else {  
            fmt.Println(err)  
        }  
    }  
    return err  
}
```

Gambar 3.12 Fungsi *periksaPenyakit*

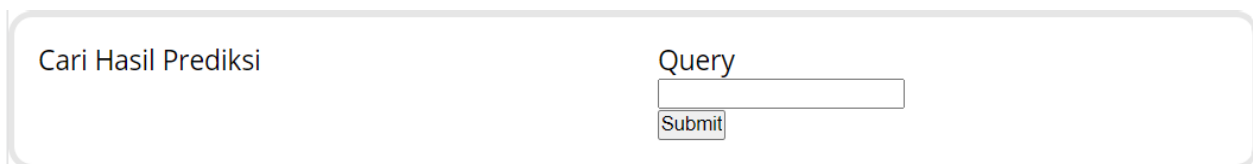
Pada penjelasan sebelumnya fungsi *periksaPenyakit* memanggil fungsi *KMPMod*, fungsi *KMPMod* dan fungsi *kmpModAlgo*. Kedua fungsi tersebut adalah fungsi yang menerapkan algoritma *pattern matching knuth-morris-pratt*. Skema dari fungsi *kmpModAlgo* akan dijelaskan lebih detail pada bab 4.

Dari sisi *frontend* telah dibuat sebuah fungsi *handlesubmit*, yang akan melakukan *post request* kepada sisi *backend* ketika tombol *submit* yang tersedia ditekan, yang kurang lebih fungsi nya sama dengan fungsi *handlesubmit* yang terdapat pada gambar 3.7. Data yang di *post* adalah data yang sebelumnya telah diisikan oleh pengguna pada *form* yang tersedia, lihat gambar 3.8. Selain fungsi *handlesubmit*, pada bagian *frontend* juga dibuat fungsi *getResult* yang akan melakukan *get request* terhadap data hasil prediksi yang baru saja di submit.

```
getResult = () => {  
  
  axios.get(Endpoints.hasilPrediksi)  
    .then(res => {  
      const records = res.data;  
      const record = records[records.length-1]  
      console.log(record)  
      this.setState({  
        results : record,  
        doneProcess : true  
      });  
    })  
}
```

Gambar 3.16 Fungsi getResult

### C. Langkah Penyelesaian Fitur Pencarian Hasil Prediksi



Gambar 3.17 Tampilan Fitur Cari Hasil Prediksi

Fitur pencarian hasil prediksi diselesaikan dengan beberapa langkah baik dari sisi *frontend* maupun dari sisi *backend*. Lingkungan implementasi web aplikasi DNA *pattern matching* menggunakan bahasa pemrograman Go pada sisi *backend*, menggunakan framework *React JS* pada sisi *frontend*, dan menggunakan DBMS Postgresql untuk sisi basis data nya.

Pertama dari sisi *backend* dibuat sebuah struct yang diberi nama `Query`, struct `Query` memiliki atribut `IdQuery` yang bertipe `int64`, dan `SearchQuery` yang bertipe `string`.

```
type Query struct {  
  IdQuery    int64 `json:"id"`  
  SearchQuery string `json:"query"`  
}
```

Gambar 3.17 Struct Query

Pada fitur ini juga digunakan struct `hasil_prediksi` yang sama dengan yang terdapat pada gambar 3.9. Pada fitur ini juga dibuat beberapa fungsi sebagai berikut :

1. fungsi *postQuery*

Fungsi ini digunakan agar sisi *frontend* dapat mengirimkan query kepada sisi *backend*. Query yang diterima nantinya akan diperiksa di sisi *backend* menggunakan fungsi yang mengimplementasikan *regular expression*, apakah format query tersebut mengikuti format tanggal yang telah ditentukan (YYYY-MM-DD) atau tidak.

```
func postQuery(c *gin.Context) {
    var newQuery Query

    if err := c.BindJSON(&newQuery); err != nil {
        return
    }

    query = append(query, newQuery)
    c.IndentedJSON(http.StatusCreated, newQuery)
}
```

Gambar 3.18 fungsi *postQuery*

## 2. fungsi *getData*

Fungsi ini digunakan agar sisi *frontend* dapat menerima data berdasarkan *input* query yang telah diterima. Query akan diperiksa terlebih dahulu menggunakan fungsi yang mengimplementasikan *regular expression*, apakah query tersebut mengikuti format tanggal yang telah ditentukan (YYYY-MM-DD). Apabila iya maka data hasil prediksi akan diambil dari basis data menggunakan fungsi *ViewHasilPrediksiByDate*, apabila tidak maka program akan mengasumsikan query yang diinput adalah nama penyakit dan data hasil prediksi akan diambil dari basis data menggunakan fungsi *ViewHasilPrediksiByName*.

```
func IsDate(s string) (bool, error) {
    var isDate bool
    var err error

    // Gak menerima string kosong
    if s == "" {
        return false, nil
    }

    isDate, err = regexp.MatchString("\\d{4}-\\d{2}-\\d{2}", s)
    return isDate, err
}
```

Gambar 3.19 Fungsi *isDate*

### 3. fungsi *ViewHasilPrediksiByDate*

Fungsi ini digunakan untuk mendapatkan data hasil prediksi dari basis data berdasarkan tanggal.

### 4. fungsi *ViewHasilPrediksiByName*

Fungsi ini digunakan untuk mendapatkan data hasil prediksi dari basis data berdasarkan nama penyakit.

### 5. fungsi *ViewHasilPrediksiByNameNDate*

Fungsi ini digunakan untuk mendapatkan data hasil prediksi dari basis data berdasarkan tanggal dan nama penyakit.

Dari sisi *frontend* telah dibuat sebuah fungsi *handlesubmit*, yang akan melakukan *post request* kepada sisi *backend* ketika tombol *submit* yang tersedia ditekan, yang kurang lebih fungsi nya sama dengan fungsi *handlesubmit* yang terdapat pada gambar 3.7. Data yang di *post* adalah data yang sebelumnya telah diisikan oleh pengguna pada *form* yang tersedia, lihat gambar 3.17. Selain fungsi *handlesubmit*, pada bagian *frontend* juga dibuat fungsi *getResult* yang akan melakukan *get request* terhadap data hasil prediksi yang diinginkan, fungsi ini kurang lebih sama dengan fungsi yang terdapat pada gambar 3.16.

## D. Fitur Aplikasi dan Arsitektur Web Aplikasi

### 1. Fitur Fungsional

- Fitur Menambahkan Jenis Penyakit
- Fitur Prediksi Penyakit
- Fitur Pencarian Hasil Prediksi berdasarkan tanggal prediksi
- Fitur Pencarian Hasil Prediksi berdasarkan nama penyakit
- Fitur Pencarian Hasil Prediksi berdasarkan tanggal dan nama penyakit

### 2. Arsitektur Web Aplikasi

- Frontend

Frontend adalah istilah yang digunakan untuk komponen website yang berada pada "client-side". Frontend mengatur bagaimana client berinteraksi dengan website. Hal yang diatur oleh front end adalah tampilan visual pada layar dan interaksi client dengan

objek pada layar. Framework yang digunakan pada proyek ini adalah ReactJS dengan bahasa pemrograman *Javascript*.

- Backend

Backend adalah istilah yang digunakan untuk komponen website yang berada pada "server-side". Backend mengatur bagaimana data diolah dan disimpan. Pada program ini, backend berguna untuk memproses pattern matching pada DNA, sanitasi rantai DNA, menyimpan data hasil pemrosesan pattern matching, dan penanganan basis data. Bahasa yang digunakan pada proyek ini adalah Go dan basis data yang digunakan adalah Postgresql.

## BAB IV

### Implementasi dan Pengujian

#### A. Spesifikasi Teknis Program

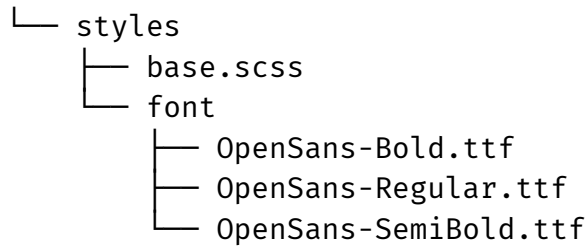
Program dapat dibagi menjadi dua bagian, bagian *frontend* dan bagian *backend*.

##### 1. Front End

Bagian *frontend* berguna sebagai cara pengguna untuk mengakses fitur-fitur dari aplikasi. Bagian ini dibuat menggunakan *Framework React*. Berikut merupakan susunan file di *frontend*:

```
.
├── README.md
├── package-lock.json
├── package.json
├── public
│   ├── favicon.ico
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── Api.js
    ├── App.js
    ├── component
    │   ├── AddPenyakit.jsx
    │   ├── AddPenyakit.scss
    │   ├── HasilPrediksi.jsx
    │   ├── HasilPrediksi.scss
    │   ├── Navbar
    │   │   ├── Navbar.jsx
    │   │   └── Navbar.scss
    │   ├── TestDNA.jsx
    │   └── TestDNA.scss
    ├── index.js
    └── pages
        ├── About.jsx
        ├── About.scss
        ├── MatcherDNA.jsx
        └── MatcherDNA.scss
```

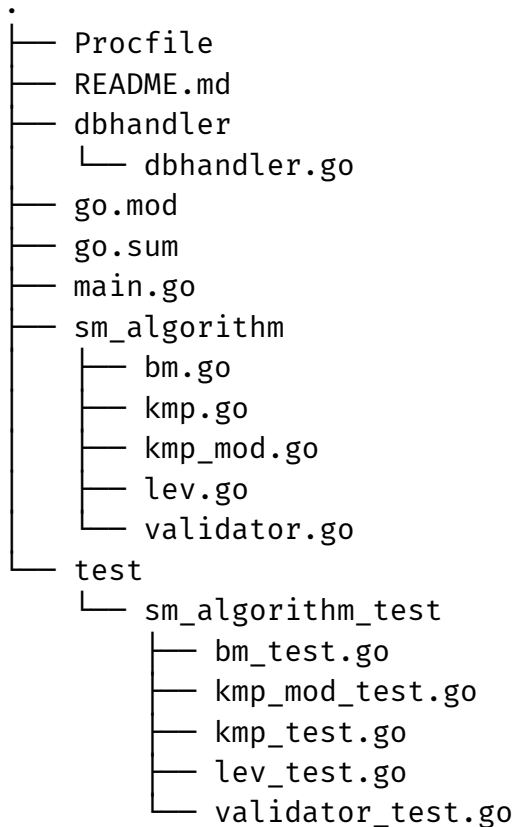




Implementasi *frontend* berada di `src/index.js`. Di folder `src` terdapat folder *Components*, yang menyimpan komponen-komponen yang dipakai di `index.js`.

## 2. Back End

Bagian *backend* berguna untuk melakukan komputasi yang dibutuhkan oleh aplikasi. Bagian ini dibuat menggunakan bahasa pemrograman Go. Susunan file di *backend* adalah sebagai berikut:



Fungsi utama *backend* ada di `main.go` dan file-file di `db_handler` dan `sm_algorithm` merupakan implementasi dari fitur-fitur yang digunakan aplikasi. Folder `sm_algorithm` berisi file berbahasa Go dengan package `sm_algorithm` yang berguna untuk menampung fitur-fitur pencocokan string menggunakan algoritma string matching ataupun regex. Folder `db_handler` berisi fungsi-fungsi yang digunakan untuk berkomunikasi dengan

*database*. Di *backend* juga terdapat folder *test*, yang berguna untuk menampung *unit test* yang dites untuk setiap algoritma.

Berikut merupakan fungsi-fungsi publik yang ada di dalam package *sm\_algorithm*.

Nama File	Nama Fungsi	Deskripsi Fungsi
bm.go	BM	Implementasi dari algoritma Boyer-Moore. Me-return (int, error) yaitu index ditemukannya string yang sesuai dan detail error pada program.
kmp.go	KMP	Implementasi dari algoritma Knuth-Morris-Pratt. Me-return (int, error) yaitu index ditemukannya string yang sesuai dan detail error pada program.
kmp_mod.go	KMPMod	Modifikasi dari algoritma Knuth-Morris-Pratt. Bedanya algoritma adalah karena ini akan menoleransi ketidaksamaan dengan jumlah yang telah ditentukan. Me-return (int, error) yaitu index ditemukannya string yang sesuai dan detail error pada program.
lev.go	LevDist	Implementasi dari algoritma Levenshtein Distance. Tidak digunakan pada program karena adanya KMPMod.
validator.go	IsValid	Mengecek apakah suatu string merupakan string DNA yang valid.
	IsDate	Mengecek apakah suatu string mengikuti format "YYYY-MM-DD".

## 2.1 Algoritma KMPMod

KMPMod merupakan algoritma hasil modifikasi dari algoritma KMP. Pada Algoritma KMP, Ketika ditemukan kesalahan, KMP akan langsung pindah. Namun, di algoritma ini, KMPMod hanya pindah ketika jumlah kesalahan melebihi batas toleransi. Ketika kesalahan melebihi batas toleransi, KMPMod akan pindah ke substring di kesalahan pertama. Berikut source code dari Algoritma tersebut:

```
func kmpModAlgo(text string, pattern string, tolerance int) (int, float64) {
    borderArray := kmpBorderArray(pattern) // Fungsi Border
```

```
textLen := len(text)
ptrnLen := len(pattern)

i, j, k, diffCount, minCount, minI := 0, 0, -1, 0, ptrnLen, -1
// i          variabel index untuk text
// j          variabel index untuk pattern
// k          kemunculan pertama perbedaan
// diffCount  jumlah perbedaan
// minCount   jumlah perbedaan paling sedikit
// minI       index text untuk minCount

for i < textLen && !(j == ptrnLen-1 && diffCount == 0) {
    for i < textLen && j < ptrnLen && diffCount <= tolerance {
        if text[i] != pattern[j] {
            diffCount++
            if k == -1 {
                k = j
            }
        }
        i++
        j++
    }
    if j == ptrnLen && diffCount < minCount {
        minCount = diffCount
        if diffCount <= tolerance {
            minI = i - j
        }
    }
    if i < textLen {
        // Penambahan berlebihan di akhir loop sebelumnya
        // sehingga dikurangi
        i--
        j--
        if diffCount > 0 {
            // Pergi ke perbedaan pertama dengan cara
            // yang sama seperti jika ada kesalahan
            // di algoritma KMP biasa
            if k > 0 {
                i = i - (j - k)
                j = borderArray[k-1]
            } else if j > 0 {
                j = borderArray[j-1]
            } else {
                i++
            }
            // Reset k dan diffCount
            k = -1
        }
    }
}
```

```
        diffCount = 0
    }
}
}
return minI, 1 - float64(minCount)/float64(ptrnLen)
}
```

## 2.2 Algoritma KMP

Algoritma KMP merupakan salah satu algoritma yang digunakan untuk pencocokan string. Algoritma ini bekerja dengan cara mencari pergeseran terbanyak sehingga menghindari perbandingan yang sia-sia. Algoritma ini bagus ketika jumlah alfabet yang digunakan kecil. Algoritma ini memiliki kompleksitas  $O(m + n)$  dengan  $m$  panjang pattern dan  $n$  panjang string di kasus rata-rata. Berikut merupakan implementasi algoritma KMP:

```
func kmpAlgo(text string, pattern string) int {
    borderArray := kmpBorderArray(pattern) // Fungsi Border

    textLen := len(text)
    ptrnLen := len(pattern)

    i, j := 0, 0

    for i < textLen {
        if text[i] == pattern[j] {
            if j == ptrnLen-1 {
                return i - (ptrnLen - 1)
            }
            i++
            j++
        } else if j > 0 {
            j = borderArray[j-1]
        } else {
            i++
        }
    }
    return -1
}
```

## 2.3 Algoritma BM

Algoritma BM merupakan salah satu algoritma yang digunakan untuk pencocokan string. Algoritma ini didasari dua teknik, yaitu teknik *looking glass* dan teknik *character jump*. Ketika terjadi ketidakcocokan, salah satu dari tiga kasus terjadi, yaitu:

- Jika pattern memiliki huruf yang sama di sebelah kirinya, maka shift kanan ke karakter tersebut.
- Jika pattern memiliki huruf yang sama tetapi shift ke kanan tidak memungkinkan, shift ke kanan sekali.
- Jika kedua kasus diatas tidak bisa, align pattern dengan karakter setelah kesalahan.

Berikut merupakan source code algoritma BM:

```
func bmAlgo(text string, pattern string) int {
    // menggunakan map, bukan array of all char
    lastOccur := bmLastOccur(pattern)

    textLen := len(text)
    ptrnLen := len(pattern)

    i := ptrnLen - 1

    if i > textLen-1 {
        return -1
    }

    j := ptrnLen - 1

    // Simulasi do while loop
    var lastOcc int
    var exists bool
    firstItr := true
    for firstItr || i <= textLen-1 {
        firstItr = false
        if pattern[j] == text[i] {
            if j == 0 {
                return i
            }
            i--
            j--
        } else {
            // Cari karakter di map
            lastOcc, exists = lastOccur[text[i]]
            if !exists {
                lastOcc = -1
            }
            i = i + ptrnLen - Mins(j, 1+lastOcc)
            j = ptrnLen - 1
        }
    }
}
```

```
    return -1  
}
```

## 2.4 Regular Expression

Aplikasi menggunakan Regular Expression untuk mengecek apakah string merupakan DNA yang valid dan apakah string merupakan string tanggal. Regular Expression di aplikasi menggunakan library regexp. Berikut merupakan fungsi yang menggunakan Regular Expression:

```
func IsValid(s string) (bool, error) {  
    var dnaSanitized bool  
    var err error  
  
    // Gak menerima string kosong  
    if s == "" {  
        return false, nil  
    }  
  
    // Apakah ada yang bukan ACGT  
    dnaSanitized, err = regexp.MatchString("[^ACGT]", s)  
    return (!dnaSanitized), err  
}  
  
func IsDate(s string) (bool, error) {  
    var isDate bool  
    var err error  
  
    // Gak menerima string kosong  
    if s == "" {  
        return false, nil  
    }  
  
    isDate, err = regexp.MatchString("\\d{4}-\\d{2}-\\d{2}", s)  
    return isDate, err  
}
```

## B. Tata Cara Penggunaan Program

Front End dari program telah di-deploy pada <https://tubes3-dna-matcher.netlify.app/>. Untuk menjalankan di lokal, lakukan

```
npm install  
npm start
```

Back End dari program juga telah di-deploy di Heroku. API Backend program dapat diakses dengan <https://tubes3-dna-matcher.herokuapp.com/>. Untuk menjalankan lokal, lakukan

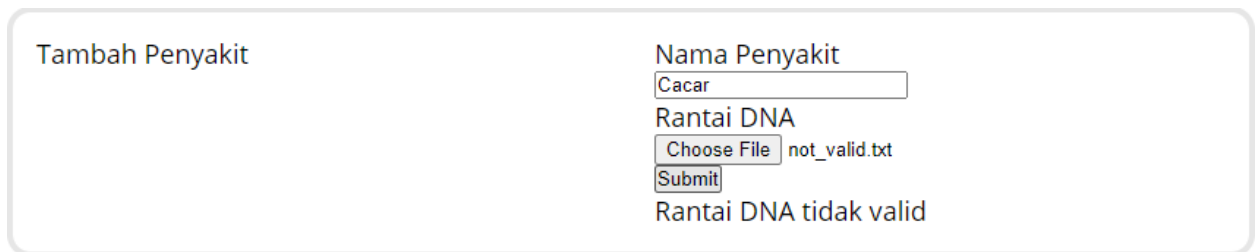
```
go build
```

```
// Link-nya ganti-ganti, sehingga bisa saja  
// URL ini tidak berfungsi  
export  
DATABASE_URL=postgres://oyukhvnsxohoku:4e4319499fb3816f3d229893806cbfd  
e848caa6550e31798bd2895be581f9bf3@ec2-34-207-12-160.compute-1.amazonaw  
s.com:5432/de4m84cglucnno
```

```
./dna-matcher
```

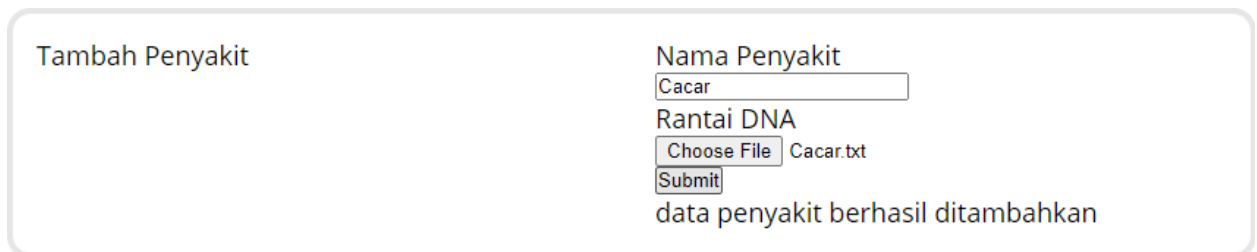
## C. Hasil Pengujian

### 1. Fitur Tambah Jenis Penyakit



The screenshot shows a web form titled "Tambah Penyakit". It has two input fields: "Nama Penyakit" with the value "Cacar" and "Rantai DNA". Below the "Rantai DNA" field, there is a "Choose File" button with the text "not\_valid.txt" next to it, and a "Submit" button. Below the "Submit" button, the message "Rantai DNA tidak valid" is displayed.

Gambar 4.1 Fitur Tambah Penyakit dengan Rantai DNA tidak valid



The screenshot shows the same "Tambah Penyakit" form. The "Nama Penyakit" field still has "Cacar". The "Rantai DNA" field is empty. The "Choose File" button now has the text "Cacar.txt" next to it. The "Submit" button is visible. Below the "Submit" button, the message "data penyakit berhasil ditambahkan" is displayed.

Gambar 4.2 Fitur Tambah Penyakit dengan Rantai DNA valid

asdfasdjfhskalakhdfjd

Gambar 4.3 not\_valid.txt

ACTGATCGATCGATGCATCGTA

Gambar 4.4 Cacar.txt

### 2. Fitur Prediksi Penyakit

Periksa Rantai DNA

Nama Pengguna  
Zian

Rantai DNA  
Choose File not\_valid.txt

Nama Penyakit  
Cacar

Submit

Rantai DNA tidak valid !

Gambar 4.5 Fitur Prediksi Penyakit dengan Rantai DNA tidak valid

Periksa Rantai DNA

Nama Pengguna  
Zian

Rantai DNA  
Choose File Cacar.txt

Nama Penyakit  
Malaria

Submit

Data penyakit tidak ditemukan !

Gambar 4.6 Fitur Prediksi Penyakit dengan Data Penyakit tidak ada

Periksa Rantai DNA

Nama Pengguna  
Zian

Rantai DNA  
Choose File Cacar\_zian.txt

Nama Penyakit  
Cacar

Submit

2022-04-28  
Zian  
Cacar  
86%  
true

Gambar 4.7 Fitur Prediksi Penyakit dengan Data yang valid

asdfasdjfhskalakhfjd

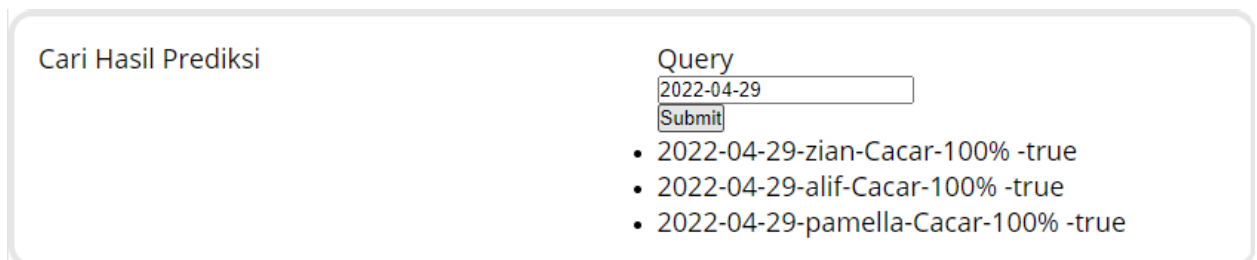
Gambar 4.8 not\_valid.txt

ACTGATCGATCGATGCATCAACGATCGTACGCTAGCT

Gambar 4.9 Cacar\_zian.txt



### 3. Fitur Pencarian Hasil Prediksi

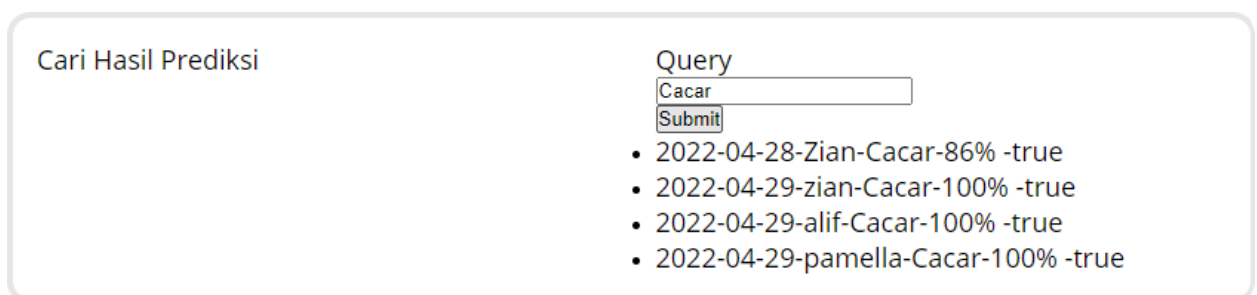


Cari Hasil Prediksi

Query  
2022-04-29  
Submit

- 2022-04-29-zian-Cacar-100% -true
- 2022-04-29-alif-Cacar-100% -true
- 2022-04-29-pamella-Cacar-100% -true

Gambar 4.10 Fitur Pencarian Hasil Prediksi dengan Input Tanggal

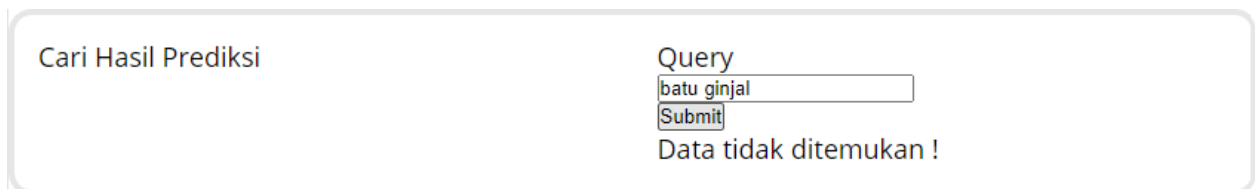


Cari Hasil Prediksi

Query  
Cacar  
Submit

- 2022-04-28-Zian-Cacar-86% -true
- 2022-04-29-zian-Cacar-100% -true
- 2022-04-29-alif-Cacar-100% -true
- 2022-04-29-pamella-Cacar-100% -true

Gambar 4.11 Fitur Pencarian Hasil Prediksi dengan Input Nama Penyakit

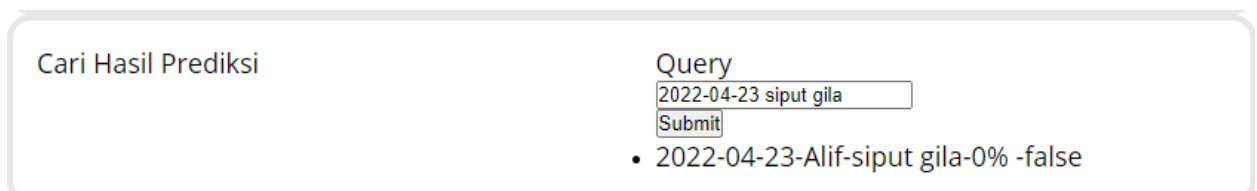


Cari Hasil Prediksi

Query  
batu ginjal  
Submit

Data tidak ditemukan !

Gambar 4.12 Fitur Pencarian Hasil Prediksi Input Data Tidak Tersedia



Cari Hasil Prediksi

Query  
2022-04-23 siput gila  
Submit

- 2022-04-23-Alif-siput gila-0% -false

Gambar 4.13 Fitur Pencarian Hasil Prediksi dengan Tanggal Nama Penyakit

### D. Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang telah dilakukan dapat dilihat bahwa semua fitur pada web aplikasi dapat berjalan dengan baik. Pada fitur penambahan jenis penyakit program berhasil memeriksa apakah rantai DNA yang telah diinput merupakan rantai DNA yang valid atau tidak.

Selain itu pada fitur prediksi penyakit juga program berhasil menghasilkan output yang sesuai harapan disertai dengan tingkat kemiripan. Pada fitur prediksi penyakit program berhasil memeriksa apakah rantai DNA merupakan rantai DNA yang valid, kemudian program juga berhasil memeriksa apakah nama penyakit yang diinput berada pada basis data.

Hasil pengujian pada fitur pencarian hasil prediksi juga menunjukkan bahwa fitur dapat berjalan dengan baik. Program berhasil menampilkan hasil prediksi berdasarkan input query tanggal prediksi saja, input nama penyakit saja, dan input tanggal dan nama penyakit dengan memeriksa terlebih dahulu input query tersebut menggunakan *regular expression*.

## BAB V

### Kesimpulan dan Saran

Berdasarkan hasil implementasi program penerapan *string matching* dan *regular expression* dalam DNA *pattern matching* dalam bentuk aplikasi web, didapatkan beberapa kesimpulan sebagai berikut :

1. Algoritma *pattern matching knuth-morris-pratt* dan *boyer-moore* berhasil diimplementasikan dengan baik pada sisi backend dengan bahasa pemrograman *Go*.
2. Program sanitasi terhadap rantai DNA menggunakan *regular expression* berhasil diimplementasikan dengan baik pada sisi backend dengan bahasa pemrograman *Go*.
3. Fitur tambah jenis penyakit berhasil berjalan dengan baik sesuai dengan spesifikasi yang ada.
4. Fitur prediksi penyakit berhasil berjalan dengan baik sesuai dengan spesifikasi yang ada.
5. Fitur pencarian hasil prediksi berdasarkan tanggal prediksi berhasil berjalan dengan baik sesuai dengan spesifikasi yang ada memanfaatkan *regular expression*.
6. Fitur pencarian hasil prediksi berdasarkan nama penyakit berhasil berjalan dengan baik sesuai dengan spesifikasi.
7. Fitur pencarian hasil prediksi berdasarkan tanggal dan nama penyakit berhasil berjalan dengan baik sesuai dengan spesifikasi.
8. Algoritma *pattern matching KMP* dan *BM* berhasil diimplementasikan dengan memberikan persentase tingkat kemiripan (bonus berhasil dikerjakan).
9. Web aplikasi baik dari sisi *frontend* maupun *backend* berhasil di deploy (bonus berhasil dikerjakan).
10. Penggunaan basis data Postgresql pada web aplikasi berhasil diimplementasikan dengan baik sesuai dengan spesifikasi yang ada.

Link Repository Github	: <a href="https://github.com/malifpy/testDNA_WebApp">https://github.com/malifpy/testDNA_WebApp</a>
Link API Backend	: <a href="https://tubes3-dna-matcher.herokuapp.com/">https://tubes3-dna-matcher.herokuapp.com/</a>
Link Frontend	: <a href="https://tubes3-dna-matcher.netlify.app/">https://tubes3-dna-matcher.netlify.app/</a>
Link Video Demo	:

Berdasarkan hasil implementasi program penerapan *string matching* dan *regular expression* dalam DNA *pattern matching* dalam bentuk aplikasi web, didapatkan beberapa saran sebagai berikut :

1. Dibutuhkan waktu untuk mempelajari bahasa pemrograman *Go*.
2. Dibutuhkan pemahaman yang baik terhadap algoritma *pattern matching KMP*, *BM*, dan *Regex* dalam implementasi program ini.

## **DAFTAR PUSTAKA**

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

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

<https://dasarpemrogramangolang.novalagung.com/1-berkenalan-dengan-golang.html>