**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

# IMPLEMENTASI ALGORITMA STRING MATCHING UNTUK MENGIDENTIFIKASI KATA/KALIMAT DALAM JUZ 30 BERBASIS ANDROID

**Hanifah Ekawati[1)], Pitrasacha Aditya[2)], Fariz Mirza Ahmad[3)]**

[1]Manajemen Informatika, [2,3] Teknik Informatika, STMIK Widya Cipta Dharma
[1,2,3]Jl.Prof M.Yamin No.25 Samarinda Kalimantan Timur 75123
E-mail: hanifah@wicida.ac.id[1)], pitra@wicida.ac.id[2)], farizmirza199@gmail.com[3)]

## ABSTRAK

Aplikasi ini bertujuan untuk mengidentifikasi kata/kalimat dalam juz 30 berbasis Android yang dapat membantu kaum muslimin untuk proses menghafal surah dalam Al-Qur'an terutama juz 30. Desain Sistem dari Aplikasi untuk Mengidentifikasi Kata/Kalimat Dalam Juz 30 Menggunakan Algoritma String Matching Berbasis Android ini menggunakan Unifield Modeling Language (UML) yang terdiri dari Use Case Diagram, Activity Diagram, Sequence Diagram, dan Class Diagram. Metode pengembangan sistem yang digunakan yaitu waterfall. Pengujian Sistem yang dilakukan yaitu dengan menggunakan Pengujian Black Box.Penelitian ini menghasilkan Aplikasi untuk Mengidentifikasi kata/kalimat dalam Juz 30 Menggunakan Algoritma String Matching berbasis Android ini diharapkan Agar dapat memberikan manfaat kepada pihak-pihak yang terkait, agar dapat mempermudah mencari suatu surah yang sedang dilantunkan oleh seseorang.

***Kata Kunci:*** *Aplikasi, identifikasi, juz amma, algoritma String Matching, Android*

## 1. PENDAHULUAN

Bagi seorang muslim, mempelajari Al-Qur'an adalah suatu kewajiban. Dalam proses belajar Al-Qur'an, biasanya diawali dengan menghafalkan lafal dari seseorang yang sedang membacakan Al-Qur'an. Setelah seseorang lancar dalam menghafalkan lafal-nya barulah akan dilanjutkan pada proses mempelajari isi serta kandungannya. Akan tetapi, dalam proses menghafal ini akan ada masalah, yaitu lupa. Ketika seseorang mendengarkan suatu bacaan Al-Qur'an, dia merasa familiar dengan bacaan tersebut namun dia tidak bisa mengingat nama surah dari bacaan tersebut.

Al-Qur'an terdiri dari 114 surah, 30 juz dan 6.236 ayat menurut riwayat Hafsh, 6262 ayat menurut riwayat Ad-Dur, dan 6214 ayat menurut riwayat Warsy. Imam Syafi'i mencatat ada 1.027.000 (satu juta dua puluh tujuh ribu) huruf dalam Al-Qur'an. Dalam juz 30 terdiri dari 37 surah, 564 ayat, dan 2308 kata. Salah satu ciri dalam penerapan teknologi adalah bahwa teknologi dapat membantu dalam memecahkan suatu permasalahan secara efektif dan efisien bagi penggunanya. Dalam perkembangannya khususnya dalam bidang teknologi informasi telah terbukti dengan diterapkanya teknologi informasi dapat membantu memecahkan berbagai permasalahan yang terjadi. Adapun bidang yang telah menerapkan teknologi khususnya teknologi informasi baik itu bidang Politik, Ekonomi, sosial, Budaya serta bidang pertahanan dan keamanan suatu negara. Bahkan bisa dikatakan salah satu ciri sebuah negara maju tercermin dalam seberapa besar tingkat penerapan teknologi khususnya Teknologi informasi. Hal inilah salah satu bukti bahwa penerapan teknologi merupakan salah satu cara dalam menyelesaikan permasalahan secara efektif dan efisien.

Maka akan dibangun sebuah aplikasi yang mampu mengidentifikasi nama surah dan ayat yang sedang dibacakan dengan data masukan berupa suara yang nantinya suara tersebut dirubah kedalam sebuah teks kemudian dicocokkan menggunakan Algoritma String Matching.

Aplikasi ini dibuat agar dapat mengetahui surah yang sedang dilantunkan.

## 2. RUANG LINGKUP

### 2.1 Cakupan Permasalahan

Adanya kebutuhan user untuk mempermudah mempelajari al quran salah satunya dengan cara mengidentifikasi nama surah dan ayat yang sedang dibacakan, sehingga mempermudah user memahami dan membaca Al Quran.

### 2.2 Batasan-batasan Penelitian

Agar pembahasan tidak menyimpang dan penelitian yang dihasilkan optimal, maka ditentukan batasan masalah sebagai berikut:

1. Aplikasi ini dioperasikan hanya dalam *smartphone* yang menggunakan OS (*Operating System*) Android.
2. Aplikasi ditujukan untuk *smartphone* dengan minimal versi Android 5
3. Aplikasi ini menggunakan sistem *online*.

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

4. Untuk pencarian data aplikasi ini menggunakan *String Matching* dengan menggunakan alur *Knuth Morris Pratt*.
5. Aplikasi ini membantu pengguna untuk mencari surah yang ingin diketahui.
6. Aplikasi ini hanya mengidentifikasi surah–surah yang berada di juz 30.
7. Aplikasi ini belum bisa mengidentifikasi ayat secara spesifik.
8. Aplikasi ini mengidentifikasi surah berdasarkan masukan berupa kata yang akan dicari.

### 2.3 Rencana Hasil yang diharapkan

1. Memudahkan umat muslim untuk mengetahui surah–surah di juz 30
2. Dengan mengetahui surah - surah yang telah diidentifikasi maka umat muslim lebih mudah untuk bisa menghafal Al-Qur'an.

## 3. BAHAN DAN METODE

Adapun bahan dan metode yang digunakan dalam membangun aplikasi ini yaitu :

### 3.1 Aplikasi

Aplikasi adalah program siap pakai yang dapat digunakan untuk menjalankan perintah-perintah dari pengguna aplikasi tersebut dengan tujuan mendapatkan hasil yang lebih akurat sesuai dengan tujuan pembuatan aplikasi tersebut (Gunawan, 2019)

### 3.2 Algoritma *String Matching*

*String matching* atau pencocokan *string* adalah algoritma yang digunakan untuk melakukan pencarian suatu string yang disebut pattern dalam string yang disebut teks.(Aan STMIK Budi Darma & Buulolo, 2016)
Algoritma *string matching* mempunyai tiga komponen utama(Matondang, 2018), yaitu :

1. *Pattern*, yaitu deretan karakter yang akan dicocokkan dengan teks, dinyatakan dengan $[0\ldots m-1]$, panjang *pattern* dinyatakan dengan $m$.
2. Teks, yaitu tempat pencocokan *pattern* dilakukan. Dinyatakan dengan $[0\ldots n-1]$, panjang teks dinyatakan dengan $n$.
3. Alfabet, berisi semua simbol yang digunakan oleh bahasa pada teks dan *pattern*, dinyatakan dengan Σ dengan ukuran dinyatakan ASIZE.

### 3.3 Algoritma String Matching Knuth Morris Pratt

Algoritma *Knuth MorrisPratt*, kita memelihara informasi yang digunakan untuk melakukan jumlah pergeseran. Algoritma KMP digunakan untuk bekerja pada arsitektur yang mendukung *string* paralel ukuran yang lebih besar(Akhtar Rasool, 2012)
Persoalan pencarian *string* dirumuskan sebagai berikut :

1. Sebuah teks (*text*), yaitu sebuah (*long*) *string* yang panjangnya n karakter.
2. *Pattern*, yaitu sebuah *string* dengan panjang m karakter (m<n) yang akan dicari dalam teks. Cari lokasi pertama di dalam teks yang bersesuaian dengan *pattern*. Aplikasi permasalahan pencocokan *string* biasa ditemukan dalam pencarian sebuah kata dalam dokumen (misalnya menu *Find* dalam *Microsoft Word*).

3. Langkah-Langkah Algoritma *Knuth Morris Pratt*

1) Mencari Fungsi Pinggiran

Algoritma *Knuth Morris Pratt* melakukan proses awal atau *preprocessing* terhadap *pattern* P dengan menghitung fungsi pinggiran (dalam literatur lain menyebut fungsi *overlap*, fungsi *failure*, dsb) yang mengindikasikan pergeseran s terbesar yang mungkin dengan menggunakan perbandingan yang dibentuk sebelum pencarian *string*. Fungsi pinggiran hanya bergantung pada karakter-karakter di dalam *pattern*, dan bukan pada karakter-karakter di dalam teks yang dicari. Oleh karena itu, dapat dilakukan perhitungan fungsi awalan sebelum pencarian *string* dilakukan.

**Tabel 1. P*seucode* algoritma KMP untuk menghitung fungsi pinggiran**

```
begin
 length ←0
  Prefix[0] ←0
            j←0
        for i ←1
   upto length step 1 do
   while j>0 & P[j+1] ≠ P[i] do
          j ←Prefix[j]
   if P[j+1] = P[i]
   then j ←j+1
   Prefix[i] ←j
 return Prefix
end
```

Keterangan :
Fungsi tersebut akan menghasilkan *output* berupa *array integer* yang merupakan angka-angka pinggiran untuk setiap posisi iterasi pada *pattern*. Barulah kemudian dapat diproses pencocokkan antara *pattern* dan teks yang diberikan.
Langkah-langkah mencari fungsi pinggiran :
(1) Menentukan deklarasi

i, j = *index*
P(j) = *pattern*
B(j) = fungsi pinggiran

(2) Menentukan *pattern* yang dicari, kemudian isi kolom *pattern.*
(3) Buat tabel fungsi pinggiran seperti contoh di bawah ini

**Tabel 2 fungsi pinggiran untuk *pattern***

| i,j | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| P(j) | | | | | |
| B(j) | | | | | |

Prefix ←

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

(4) Menambahkan *index* di atas *pattern* untuk memudahkan pergeseran *index* [*i, j*].
(5) Meletakkan posisi [*i*] berada di *index* awal [0].
(6) Meletakkan posisi [*j*] berada di *index* kedua [1].
(7) Untuk nilai awalan *prefix* pasti bernilai 0.
(8) *Pattern* [*i*] dibandingkan dengan *pattern* [*j*], jika *pattern* sama maka *prefix* [*j*] bernilai 1 jadi ( [*i*] + 1 ) dan ( [*j*] + 1 ) namun jika *pattern* [*i*] dan [*j*] tidak sama maka *prefix* [*j*] bernilai 0 dan ( [*j*] + 1) dan seterusnya hingga *j* berada di akhir *pattern*.

2) Pencocokan *String*

Kemudian cara untuk melakukan pencocokan *string* dengan menggunakan algoritma *Knuth Morris Pratt*. Menurut Kourie, Justin dalam Setiawan (2016), secara sistematis langkah-langkah yang dilakukan algoritma *Knuth Morris Pratt* pada saat mencocokkan *string* :

(1) Algoritma *Knuth Morris Pratt* mulai mencocokkan *pattern* pada awal teks.
(2) Dari kiri ke kanan, algoritma ini akan mencocokkan karakter per karakter *pattern* dengan karakter di teks yang bersesuaian, sampai salah satu kondisi berikut dipenuhi:

a. Karakter di *pattern* dan di teks yang dibandingkan tidak cocok (*mismatch*).
b. Semua karakter di *pattern* cocok. Kemudian algoritma akan memberitahukan penemuan di posisi ini.
c. Algoritma kemudian menggeser *pattern* berdasarkan tabel, lalu mengulangi langkah 2 sampai *pattern* berada di ujung teks. (Purba et al., 2021)

**Tabel 3 Pseucode Algoritma KMP pencarian *String***

```
i ←1, j ←1, k ←1
while n – k ≥ m do
while j ≤ m & T[i] = P[j] do i ← i + 1
                             j ←j + 1
if j > m then output k
if Prefix[j - 1] > 0 then  k ← i – Prefix[j - 1]
else  if i = k then i ←i + 1
                      k ←i
if j > 1 then j ← Prefix[j - 1] + 1
```

### 3.4 Metode *Waterfall*

Menurut (Shalahudd(Gunawan, 2019)in, 2014)model air terjun (*Waterfall*) sering juga disebut model sekuensial linier (*sequential linear*) atau alur hidup klasik (*classic life cycle*). Model air terjun menyediakan pendekatan alur hidup perangkat lunak secara sekuensial atau terurut. Dimulai dari analisis, desain, pengkodean, pengujian dan tahapan pendukung (*support*) atau pemeliharaan (*maintenance*). Berikut adalah tahapan-tahapan dalam metode tersebut :

1. Analisis

Proses pengumpulan kebutuhan dilakukan secara intensif untuk menspesifikasikan kebutuhan perangkat lunak agar dapat dipahami perangkat lunak apa yang dibutuhkan oleh *user*.

2. Desain

Desain perangkat lunak adalah proses multi langkah yang fokus pada desain pembuatan program perangkat lunak termasuk struktur data, dan prosedur pengkodean.

3. Pengkodean

Desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahap ini adalah program komputer sesuai dengan desain yang telah dibuat pada tahap desain.

4. Pengujian

Pengujian fokus pada perangkat lunak dari segi logik dan fungsional dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk meminimalisir kesalahan (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.

5. Pemeliharaan (*Maintenance*)

Tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketika sudah dikirimkan ke *user*. Perubahan bisa terjadi karena adanya kesalahan yang muncul dan tidak terdeteksi saat pengujian atau perangkat lunak harus beradaptasi dengan lingkungan baru. Tahap pemeliharaan dapat mengulangi proses pengembangan mulai dari analisis spesifikasi untuk perubahan perangkat lunak yang sudah ada, tetapi tidak untuk membuat perangkat lunak baru.

Dari kenyataan yang terjadi sangat jarang model air terjun dapat dilakukan sesuai alurnya karena sebab berikut :

a. Perubahan spesifikasi perangkat lunak terjadi di tengah alur pengembangan.
b. Sangat sulit bagi pelanggan untuk mendefinisikan semua spesifikasi di awal alur pengembangan.
c. Pelanggan tidak mungkin bersabar mengakomodasi perubahan yang diperlukan di akhir alur pengembangan.

Dengan berbagai kelemahan yang dimiliki model air terjun tapi model ini telah menjadi dasar dari model-model yang lain dalam melakukan perbaikan model pengembangan perangkat lunak.

Model air terjun sangat cocok digunakan kebutuhan pelanggan sudah sangat dipahami dan kemungkinan terjadinya perubahan kebutuhan selama pengembangan perangkat lunak kecil.

Hal positif dari model air terjun adalah struktur tahap pengembangan sistem jelas, dokumentasi dihasilkan disetiap tahap pengembangan, dan sebuah tahap dijalankan setelah tahap sebelumnya selesai dijalankan (tidak ada tumpang tindih pelaksanaan tahap).

## 4. PEMBAHASAN

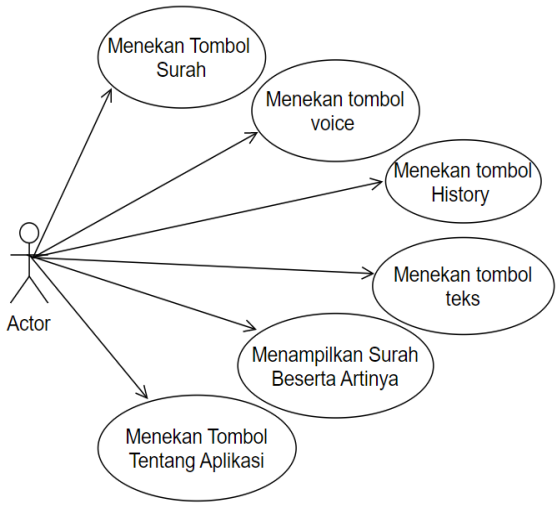
Bagian ini akan membahas mengenai desain visuakisasi, rancangan dan dokumentasi sistem dengan menggunakan alat pengembangan sistem *Unified Modelling Language* (UML), berikut desain yang digunakan oleh peneliti dengan 3 metode, yaitu *Use Case*

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

Diagram, *Activity* Diagram, *Sequence* Diagram, dan *Class* Diagram.

### 4.1 *Use Case* Diagram

Pada perancangan *use case* diagram dapat dilihat apa saja interaksi dan yang bisa dilakukan oleh *actor*.

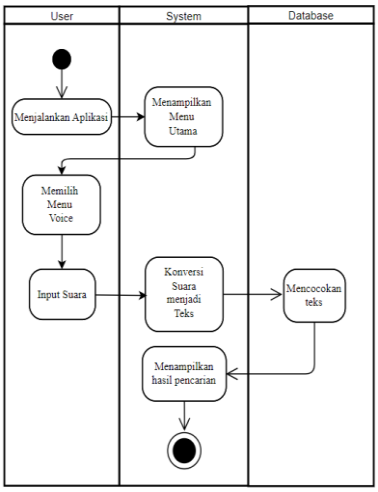



**Gambar 1. *Use Case Diagram User***

Pada gambar 1 *Use Case Diagram User* dapat dilihat bahwa prosedur rancangan sistem untuk aktor *user* dapat menekah tombol surah, menekan tombol *voice*, melihat *history* pencarian suara, menekan tombol teks, menampilkan surah beserta artinya, dan melihat tombol tenang aplikasi. Saat *User* menekan tombol surah maka program akan menampilkan detail surah yang di dalamnya berisi surah-surah yang ada pada juz amma dari an-naba surah ke 78 hingga surah an-nas yang merupakan surah ke 114 atau surah terakhir, pada detail surah juga akan menampilkan arti dari tiap ayat. Saat *User* menekan tombol *voice* maka program akan menampilkan tampilan berupa *input* suara lalu program akan merubah suara tersebut menjadi bentuk teks yang kemudian teks tersebut akan dicari menggunakan algoritma KMP, setelah itu teks yang sudah diubah tadi dicocokan dari *database*. Saat *user* menekan tombol *history* maka program akan menampilkan daftar *record* dari pencarian di tombol *voice*, jika daftar *record* tidak ada atau kosong maka *user* belum pernah melakukan pencarian melalui tombol *voice*. Saat *user* menekan tombol teks maka program akan menampilkan tampilan berupa inputan berupa teks yang ingin dicari, setelah *user* menginputkan teks yang ingin dicari maka program akan mencocokannya dari *database*, jika berhasil maka akan menampilkan teks yang diinput tadi ada pada surah dan ayat mana saja.

### 4.2 *Activity* Diagram

*Activity* Diagram menggambarkan aliran aktivitas sebuah sistem yang sedang dirancang, aliran aktivitas digambarkan dari awal sistem berjalan sampai selesai.

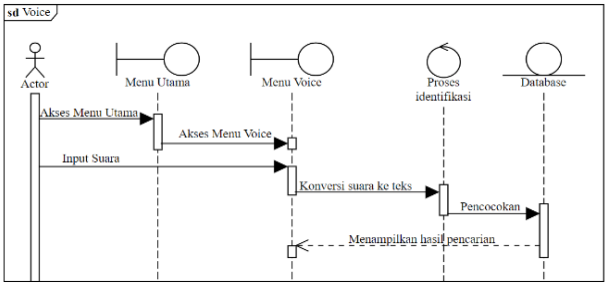



**Gambar 2. Activity Diagram Voice**

Pada gambar 2 *Activity* Diagram Menu *Voice* menggambarkan kejadian *user* memilih menu *voice* maka akan diarahkan ke halaman pencarian suara, kemudian menginputkan suara yang dilantunkan yang nantinya suara tersebut diubah menjadi teks kemudian dicocokkan dengan *database* menggunakan algoritma *String Matching* KMP dan jika cocok maka akan ditampilkan di layar pencarian dengan disertakan surah, ayat beserta artinya.

### 4.3 *Sequence* Diagram

*Sequence* Diagram menggambarkan urutan *event* dan waktu dari suatu pesan yang terjadi antar objek dalam sebuah *use case*.



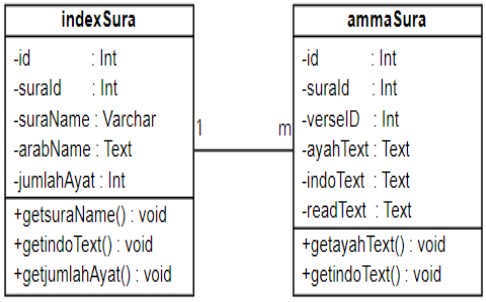

**Gambar 3 *Sequence* Diagram *Voice***

Pada gambar 3 menjelaskan urutan kejadian setelah *user* menjalankan aplikasi kemudian mengakses menu utama kemudian memilih menu *voice*. Setelah itu *user* diminta untuk *input* suara murottal untuk direkam kemudian diproses oleh *system* untuk proses identifikasi yang sekaligus dicocokan ke dalam *database* menggunakan algoritma KMP lalu tunggu beberapa saat, kemudian *system* akan menampilkan hasil pencarian.

### 4.4 *Class* Diagram

Pada *class* diagram menampilkan *class* diagram yang ada pada aplikasi android yang dihubungkan ke setiap menu maupun database lainnya dalam bentuk diagram pada gambar 4.

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**




**Gambar 4. *Class* Diagram**

*Class* indexSura memiliki atribut id, suraId, sura*Name*, arab*Name*, dan jumlahAyat. Untuk *class* ammaSura memiliki atribut id, suraId, verseID, ayah*Text*, indo*Text*, dan read*Text*.

Hasil implementasi berupa perangkat lunak (*software*) sebuah aplikasi Mengidentifikasi kata/kalimat dalam juz 30 menggunakan algoritma *String matching* berbasis android berikut :

### 4.5 Penerapan Algoritma *String Matching Knuth morris Pratt* (KMP)

Untuk tahap ini akan dijelaskan lebih lanjut tentang bagaimana algoritma *String matchingKnuth Morris Pratt (KMP)* bekerja pada aplikasi mengidentifikasi kata/kalimat dalam juz 30 yang akan dirancang, berdasarkan sistem aplikasi yang akan dibangun menggunakan algoritma KMP. Algoritma KMP memelihara informasi dan menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter.

Penerapan algoritma ini mengambil contoh kasus pencarian surah abasa, adapun langkah pencariannya setelah direkam menggunakan google *voice* yang kemudian diubah menjadi sebuah teks dan teks inilah yang akan dicocokan ke dalam *database* menggunakan algoritma *String Matching* KMP sebagai berikut :

Teks : KALAALLAHU

*Pattern* : ALLAH

Langkah pertama pecahkan *string* teks dan *string pattern* menjadi *array* kemudian menghitung fungsi pinggiran untuk *pattern* tersebut. Dapat dilihat pada tabel 4 *array* teks dan tabel 5 fungsi pinggiran.

**Tabel 4. *Array* Teks**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| T : | K | A | L | A | A | L | L | A | H | U |

**Tabel 5. Fungsi Pinggiran**

| J | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| T(j) | A | L | L | A | H |
| B(j) | 0 | 0 | 1 | 2 | 0 |

**Tabel 6. Pencocokan *Pattern***

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| T : | K | A | L | A | A | L | L | A | H | U |
| P : | A | L | L | A | H | | | | | |

Langkah ke-dua, bandingkan sisi paling kiri *pattern* (*P)* dengan sisi paling kiri teks (*T)*. Dapat dilihat pada tabel 6 di atas, terjadi ketidakcocokan karakter pada awal pencocokan *pattern*, maka dilakukan pergeseran *pattern* ke kanan sebanyak satu karakter, dapat dilihat pada tabel 7 di bawah.

**Tabel 7. Pencocokan *Pattern*2**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| T : | K | A | L | A | A | L | L | A | H | U |
| P : | | A | L | L | A | H | | | | |

Langkah ke-tiga, terlihat pada tabel 7 karakter pada posisi*i*(1) sampai*i*(2) memiliki pola yang sama, tetapi *pattern* tidak cocok dengan teks pada posisi*i*(3) Karena terjadi ketidakcocokan karakter, maka dilakukan pergeseran *pattern* ditentukan dari pinggiran awalan*P* yang bersesuaian, pada contoh diatas, awalan yang bersesuaian adalah 'AL', dengan panjang*i*=2. Nilai pinggiran terpanjang untuk *string P*[0..1] adalah*B*(1)=0. Maka jumlah pergeseran *I*-*B*=2-0=2. Jadi *pattern* digeser sebanyak 2 karakter kekanan dihitung dari awal *pattern*, terlihat pada tabel 8 di bawah.

**Tabel 8. Pencocokan *Pattern*3**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| T : | K | A | L | A | A | L | L | A | H | U |
| P : | | | | A | L | L | A | H | | |

Langkah ke-empat, melakukan kembali perbandingan karakter dari*i*(3) yaitu bandingkan ujung kiri pattern (P) dengan ujung kiri Teks (T), terlihat pada tabel 8 di atas. Karakter pada posisi*i*(3) memiliki pola yang sama, tetapi *pattern* tidak cocok dengan teks pada posisi*i*(4) Karena terjadi ketidakcocokan karakter, maka dilakukan pergeseran *pattern* kekanan sebanyak satu karakter, terlihat pada tabel 9 di bawah.

**Tabel 9. Pencocokan *Pattern*4**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| T : | K | A | L | A | A | L | L | A | H | U |
| P : | | | | | A | L | L | A | H | |

Langkah ke-lima, selanjutnya melakukan kembali perbandingan karakter dari *i*(4), yaitu bandingkan ujung kiri *pattern* (P) dengan ujung kiri teks (T) diawali dari *i*(4), terlihat pada tabel 9 di atas. Ditemukan pola *Pattern* (P) yang cocok pada karakter teks. Karena telah menemukan kecocokan *pattern* pada teks maka proses pencarian tidak diteruskan. Setelah kecocokan ditemukan

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

pada algoritma KMP maka informasi yang disimpan tadi akan ditampilkan dilayar hasil pencarian aplikasi .

### 4.6 Implementasi Algoritma *String Matching* KMP pada Aplikasi

```
function preKMP($pattern){
  $i = 0;
  $j = $lompat[0] = -1;
  while($i<count($pattern)){
    while($j>-1 && $pattern[$i]!=$pattern[$j]){
      $j = $lompat[$j];}
    $i++;
    $j++;
    if($pattern[$i]==$pattern[$j]){
      $lompat[$i]=$lompat[$j];
    }else{
      $lompat[$i]=$j;}}
  return $lompat;}
```

**Gambar 5 Implementasi PreKMP**

Pada gambar 5 menjelaskan *coding* untuk membuat fungsi pinggiran, menentukan *prefix*, menambahkan *index* di atas pattern untuk memudahkan pergeseran *index*.

```
<?php
class KMP{
  function KMPSearch($p,$t){
    $hasil = array();
    $pattern = str_split($p);
    $text    = str_split($t);
    $lompat = $this->preKMP($pattern);
  $i = $j = 0;
    $num=0;
    while($j<count($text)){
      while($i>-1 && $pattern[$i]!=$text[$j]){
        $i = $lompat[$i];}
      $i++;
      $j++;
      if($i>=count($pattern)){
        $hasil[$num++]=$j-count($pattern);
        $i = $lompat[$i];} }
  return $hasil;}
```

**Gambar 6. Implementasi Algoritma KMP *Search***

Pada gambar 6 menjelaskan mengenai pencarian KMP berupa inputan p yang berarti *pattern* dan t berupa teks. dan hasil *output* berupa hasil posisi *string* pada teks.

```
  function KMPReplace($str1,$str2,$text){
    $num = 0;
    $location = $this->KMPSearch($str1,$text);
  $t = '';
  $n = 0; $nn = 0;
  foreach($location as $in){
    $t .= substr($text,$n+$nn,$in-$n-$nn).$str2;
    $nn = strlen($str1);
    $n = $in;}
  $t .= substr($text,$n+$nn);
  return $t;}}
```

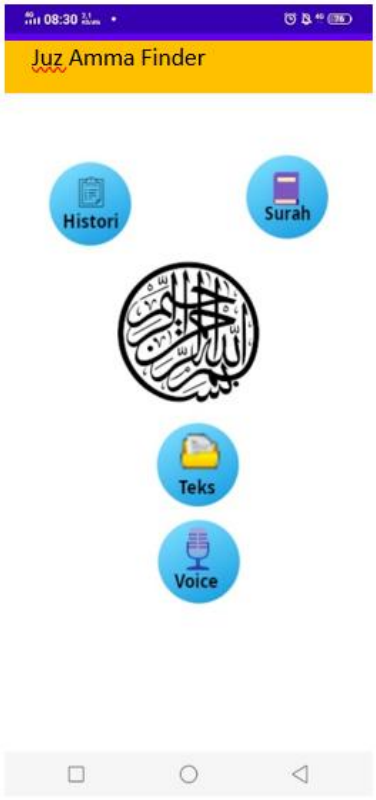
**Gambar 7. Implementasi KMP *Replace***

Pada gambar 7 merupakan implementasi algoritma KMP untuk memfilter teks yang sedang dicari, jadi str1 berfungsi sebagai string yang akan diganti dengan str2 dan str2 merupakan string yang akan mengganti str1 berdasarkan inputan text yang sedang dicari.

### 4.7 Desain Tampilan

Pada tahap ini menjelaskan pada masing-masing halaman yang terdapat pada aplikasi

1) Tampilan Menu Utama




**Gambar 8. Halaman Menu Utama**

Pada gambar 8 Halaman menu utama merupakan tampilan awal aplikasi yang berisi logo basmallah lalu ada tombol yang dapat dipilih oleh pengguna seperti *history*, surah, *voice* dan teks.

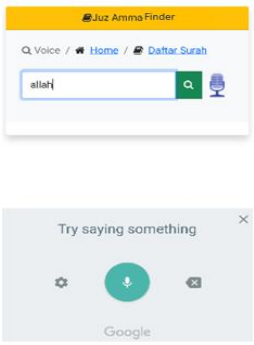
2) Tampilan Menu *History*


**Gambar 9 Tampilan Menu *History***

Pada gambar 9 merupakan tampilan tombol di menu *history*, yang berisi data hasil pencarian yang sudah dicari dari tombol *voice*.

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

3) Tampilan *Voice*




**Gambar 10. Tampilan Menu *voice***

Pada gambar 10 merupakan tampilan yang muncul ketika menekan tombol *voice*, pada tampilan ini *user* diminta untuk melantunkan surah yang ingin dicari.


**Gambar 11. Menampilkan Hasil Pencarian**

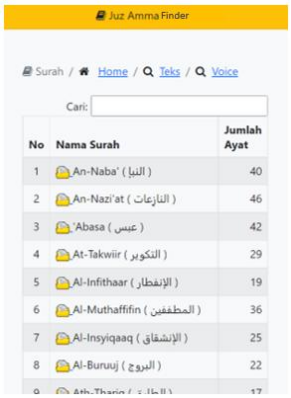
Pada gambar 11 merupakan hasil dari pencarian dari menu *voice* sebelumnya yang membutuhkan proses penginputan suara kemudian diubah menjadi teks lalu melakukan proses pencarian menggunakan algoritma KMP.

4) Tampilan Teks


**Gambar 12 Tampilan menu teks**

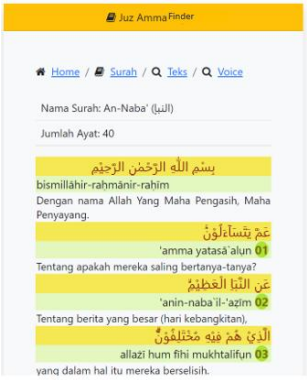
Pada gambar 12 merupakan tampilan yang muncul setelah menekan tombol teks, pada tampilan ini *user* diminta untuk memasukkan teks yang ingin dicari. Untuk teks yang dicari bukan berupa potongan ayat melainkan arti dari ayat yang ingin dicari.

5) Tampilan Surah




**Gambar 13 Tampilan Menu Surah**

Pada gambar 13 merupakan tampilan yang muncul ketika menekan tombol surah, pada tampilan ini *user* dapat memilih surah mana saja yang ingin dibaca.




**Gambar 14 Tampilan Detail Surah**

Pada gambar 14 adalah tampilan detail surah, tampilan ini muncul ketika *user* sudah memilih surah yang ingin dibaca, pada tampilan ini pula menampilkan nama surah, jumlah ayat, isi surah beserta artinya.

Menampilkan aplikasi yang dibagun, baik dalam bentuk software, hardware, jaringan komputer, dan lain-lain.

Sertakan data pendukung yang berupa desain/perancangan, tabel, grafik, gambar, atau alat penolong lain seperlunya untuk memperjelas dan mempersingkat uraian yang harus diberikan.

## 5. KESIMPULAN

Aplikasi Untuk Mengidentifikasi Kata/Kalimat Dalam Juz 30 Menggunakan Algoritma *String Matching* Berbasis Android dibangun dengan menggunakan *Google Voice*, pembuatan aplikasi ini menggunakan Android Studio sebagai *text* editornya, perancangan dan

**ISSN 2829 -1085 doi.org/10.46984/inf-wcd.2201**

pembuatan aplikasi ini menggunakan bahasa pemrograman *Java, MySQL* sebagai *databasewebsite* sementara.Aplikasi Juz Amma ini Menggunakan Algoritma *String Matching* Berbasis Android agar dapat memudahkan masyarakat dan pengguna *mobile* yang sudah mendukung sistem operasi Android untukmengidentifikasi surah-surah yang ada pada juz amma.

## 6. SARAN

Berdasarkan dari kesimpulan yang telah dikemukakan diatas, maka memberikan saran-saran sebagai berikut:

1. Aplikasi ini hanya dapat berjalan pada satu *platform* yaitu Android. Kelemahan ini menjadi acuan untuk dapat dikembangkan lagi agar dapat digunakan di beberapa *platform*.
2. Peningkatan fitur pada aplikasi Juz Amma *finder* ini masih harus ditingkatkan terutama untuk jarak yang masih terbilang cukup dekat. Aplikasi ini belum bisa mengidentifikasi ayat yang dicari secaras pesifik, untuk itu perlu ditingkatkan lagi agar bisa mengidentifikasi dengan lebih baik. Aplikasi ini masih terbatas untuk mengidentifikasi di juz 30 saja,
3. Harapan untuk kedepannya bisa ditingkatkan untuk dapat mengidentifikasi seluruh isi Al-Qur'an.

## 7. DAFTAR PUSTAKA

Aan STMIK Budi Darma, M., & Buulolo, E. (2016). *Determination of Education Scholarship Recipients Using Preference Selection Index View project Decision Support Systems View project*. https://www.researchgate.net/publication/31424523.

Akhtar Rasool, A. T. G. S. K. (2012). *String Matching Methodologies:A Comparative Analysis*. (IJCSIT) International Journal of Computer Science and Information Technologies. http://www.ijcsit.com/docs/Volume 3/Vol3Issue2/ijcsit2012030219.pdf.

Gunawan, W. (2019). Pengembangan Aplikasi Berbasis Android Untuk Pengenalan Huruf Hijaiyah. JURNAL INFORMATIKA, 66-69.

Matondang, Z. A. (2018). Implementasi Algoritma String Matching Pencarian Kata Dari Makna Rambu Lalulintas Berbasis Android. *Jurnal Sistem Informasi Kaputama (JSIK)*, *2*(1). https://doi.org/10.1234/JSIK.V2I1.93.

Purba, A. R., Ginting, G. L., & Ikhwan, I. (2021). Implementasi Algoritma String Matching Pada Pencarian Arti Istilah-Istilah Pramuka Berbasis Mobile. *Pelita Informatika: Informasi Dan Informatika*, *6*(4), 384–388. http://www.stmik-budidarma.ac.id/ejurnal/index.php/pelita/article/view/778.

Shalahuddin, M. (2014). *Rekayasa Perangkat Lunak: Terstruktur dan Berorientasi Objek*. Informatika Bandung.