# BAB III

# ANALISIS DAN PERANCANGAN SISTEM

## 3.1 Analisis Sistem

*Stemming* merupakan sebuah cara untuk menghilangkan imbuhan seperti berupa awalan dan akhiran agar didapatkan kata dasar (*root word*) (Novitasari, 2016). *Algoritma stemming* Nazief & Adriani mengembangkan berdasarkan aturan *morfologi* bahasa indonesia yaitu, mengelompokan awalan (*prefix*), sisipan (*infix*), akhiran (*suffix*), dan gabungan awalan akhiran (*confixes*). Pada *algoritma stemming* Nazief & Adriani tidak semua kata yang memiliki imbuhan bisa diselesaikan, ada beberapa kata yang mengalami pemenggalan imbuhan yang lebih (*overstemming*) atau pemenggalan imbuhan yang terlalu sedikit (*understemming*).

Maka diperlukan sebuah sistem untuk mencari kata dasar yang tepat. Kata yang mengalami pemenggalan *understemming* atau *overstemming* akan dilakukan pencarian kata dasar dengan cara dibandingkan dengan kamus kata dasar bahasa indonesia yang terdapat dalam *database*. Kata hasil perbandingan tersebut akan diurutkan berdasarkan nilai yang tertinggi. Perbandingan kata tersebut menggunakan metode *jaro winkler*.

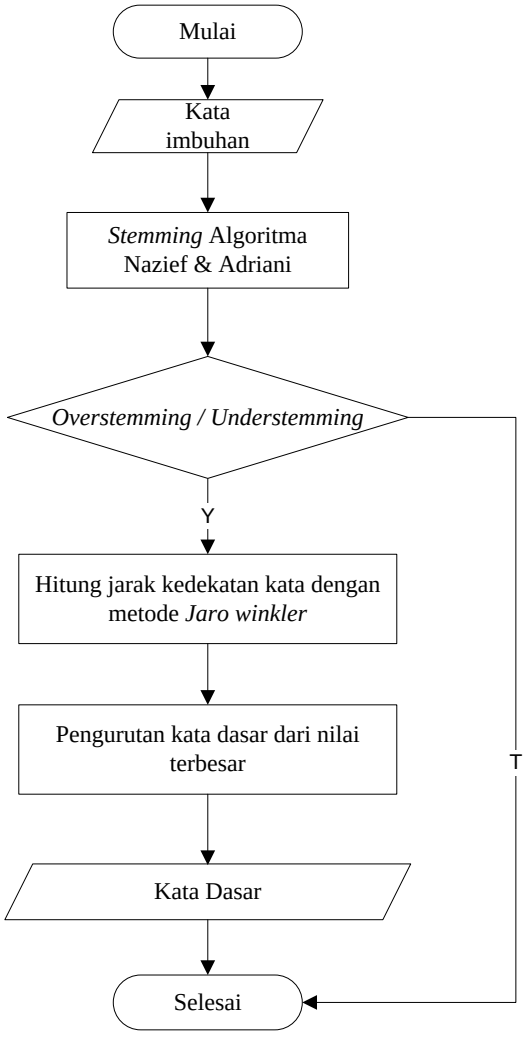
## 3.2 Hasil Analisis

Hasil analisis dari sistem modifikasi Nazief & Adriani *stemmer* untuk bahasa indonesia yang dibangun yaitu didapatkannya perubahan kata yang mengalami *overstemming* atau *understemming* pada proses *stemming* Nazief & Adriani sehingga didapatkan sebuah kata dasar menurut kamus yang ada di *database*. Modifikasi ini menggunakan metode *Jaro Winkler* untuk mengukur kesamaan kata. Dengan menggunakan metode *Jaro Winkler*, kata yang akan diukur yaitu kata yang dihasilkan oleh *algoritma stemming* Nazief & Adriani dengan kamus kata dasar yang tersimpan dalam *database*. Setelah

diukur dilakukan pengurutan kata dasar dari nilai yang tertinggi yaitu mendekati 1 atau sama dengan 1.

### 3.2.1 Deskripsi Sistem

Sistem yang akan dibangun adalah sistem yang dapat merubah kata yang telah mengalami *overstemming* dan *understemming* pada proses *stemming* Nazief & Adriani sehingga didapatkan sebuah kata dasar menurut kamus yang ada di *database*. Modifikasi ini menggunakan metode *Jaro Winkler*. Gambaran umum sistem yang akan dibangun seperti pada gambar 3.1.




**Gambar 3. 1** *Flowchart* sistem modifikasi *stemmer* Nazief & Adariani

Penjelasan Gambar 3.1 :

1. Masukkan kata inputan yang memiliki imbuhan.
2. Setelah kata dimasukkan akan dilakukan proses *stemming* menggunakan *algoritma* Nazief & Adriani.
3. Selanjutnya pengguna menganalisa kata yang dihasilkan dari proses *stemming* apakah mengalami *overstemming / understemming* atau tidak. Apabila kata yang dihasilkan terdapat *overstemming / understemming* maka dilanjutkan proses selanjutnya apabila tidak maka proses dianggap telah selesai.
4. Kata yang mengalami *overstemming / understemming* akan dihitung jarak kedekatan kata.
5. Didapatkan nilai *similarity* kata
6. Pengurutan kata dengan dimulai dengan nilai jarak kesamaan (*similarity*) yang terbesar.
7. Didapatkan kata dasar yang mendekati nilai 1 atau sama dengan 1.

## 3.3 Representasi Model

### 3.3.1 Kata Imbuhan

Kata yang diinputkan adalah kata yang memiliki imbuhan seperti awalan (*prefix*), sisipan (*infix*), akhiran (*suffix*), dan gabungan awalan akhiran (*confixes*). Seperti contoh kata Keliaran, penGecekan, MEMADAMKAN, dll.

### 3.3.2 *Stemming* Nazief & Adriani

Proses *stemming* merupakan proses pencarian kata dasar terhadap sebuah kata yang telah dilakukan proses *case folding*, *tokenizing*, dan *stopword*.

#### 3.3.2.1 *Case folding*

Tahap ini merupakan tahap dimana kata yang dimasukkan pertama diproses. *Case folding* merupakan proses penyamaan huruf dengan mengubahnya menjadi huruf kecil (*lowercase*). Pada kata yang dimasukkan terdapat huruf besar (kapital) dan kecil. Perbedaan bentuk

huruf pada kata tersebut akan mengganggu proses selanjutnya maka dari itu diperlukan proses *case folding* untuk merubah kata menjadi huruf kecil semua.

**Tabel 3.1** Contoh proses *case folding*

| **Input** | **Output** |
|---|---|
| Keliaran | keliaran |
| menGecek | mengecek |
| MEMADAMKAN | memadamkan |

#### 3.3.2.2 *Tokenizing*

Proses *tokenizing* adalah tahap pemotongan kata inputan berdasarkan kata yang menyusunnya. Pada prinsipnya proses ini adalah memisahkan setiap kata pada suatu kalimat. Setiap kata terindentifikasi mengandalkan spasi pada setiap kata akan dilakukan pemisahan kata.

**Tabel 3.2** Contoh Proses *tokenization*

| **Input** | **Output** |
|---|---|
| bertanggung jawab | bertanggung, jawab |
| air mata | air , mata |
| mengecek | mengecek |

#### 3.3.2.3 *Stopword*

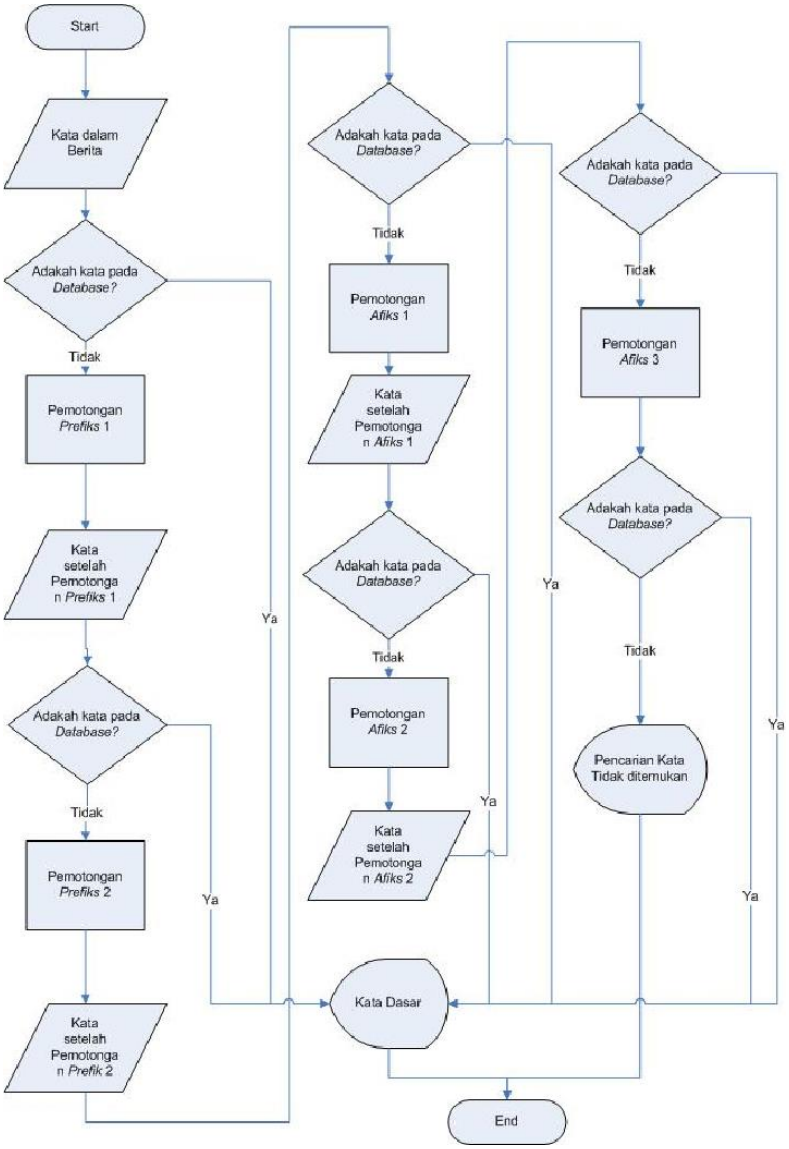
*Stopword* adalah proses menghilangkan kata-kata yang tidak memiliki arti seperti kata “yang”, “pada”, “itu” dan lain sebagainya.

**Tabel 3.3** Contoh Proses *stopword*

| **Input** | **Output** |
|---|---|
| yang | (dihapus) |
| pada | (dihapus) |
| mengecek | mengecek |

| | |
|---|---|
| mengecek | mengecek |

Pada proses ini kata akan dirubah menjadi kata dasar sehingga kata yang memiliki imbuhan seperti awalan (*prefix*), sisipan (*infix*), akhiran (*suffix*), dan gabungan awalan akhiran (*confixes*) akan dihapus dan disesuaikan kata dasarnya. Proses *stemming* menggunakan *algoritma* Nazief & Adriani. Alur proses dalam *stemming* seperti pada gambar 3.3




**Gambar 3. 2** Alur proses *stemming Nazief & Adriani*

Proses tahapan *stemming* Nazief & Adriani yang terjadi pada gambar 3.2 :

1. Cari kata yang akan di*stemming* dalam kamus kata dasar. Jika ditemukan maka diasumsikan kata adalah kata dasar. Maka *algoritma* berhenti.
2. Hilangkan *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”). Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “-nya”), jika ada.
3. Hilangkan *Derivation Suffixes* (“-i”, “-an” atau “-kan”). Jika kata ditemukan di kamus, maka *algoritma* berhenti. Jika tidak maka ke langkah 3a
    a. Jika “-an” telah dihapus dan huruf terakhir dari kata tersebut adalah “-k”, maka “-k” juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka *algoritma* berhenti. Jika tidak ditemukan maka lakukan langkah 3b.
    b. Akhiran yang dihapus (“-i”, “-an” atau “-kan”) dikembalikan, lanjut ke langkah 4.
4. Hilangkan *Derivation Prefix* (“di-”, “ke-”, “se-”, “me-”, “be”, “pe-”, “te-”) dengan iterasi maksimum sebanyak 3 kali .
    a. Langkah 4 berhenti jika :
        1. Terjadi kombinasi awalan dan akhiran.
        2. Awalan yang dideteksi saat ini sama dengan awalan yang dihilangkan sebelumnya.
        3. Tiga awalan telah dihilangkan.

        **Tabel 3.4** Kombinasi Awalan-Akhiran yang tidak diizinkan Awalan Akhiran yang tidak diizinkan

| **Awalan** | **Akhiran yang tidak diizinkan** |
|---|---|
| be- | -i |
| di- | -an |
| ke- | -i, -kan |
| me- | -an |
| se- | -i, -kan |
| te- | -an |

b. Identifikasi tipe awalan dan hilangkan. Tipe awalan ada 2 yaitu:
    1. Standart : “di-”, “ke-”, “se-” dapat langsung dihilangkan dari kata
    2. Kompleks : “me-”, “be”, “pe-”, “te-” adalah tipe awalan yang dapat bermorfologi sesuai kata dasar yang mengikutinya. Oleh karena itu, gunakan aturan pada Tabel 3.4 untuk mendapatkan pemenggalan yang tepat.
c. Cari kata yang telah dihilangkan awalannya ini di dalam kamus. Apabila tidak ditemukan, maka langkah 4 diulangi kembali. Apabila ditemukan maka keseluruhan proses selesai.

5. Apabila setelah langkah 4 kata dasar masih belum ditemukan, maka proses *recoding* dilakukan dengan mengacu pada aturan pada Tabel 2.3. *Recoding* dilakukan dengan menambahkan karakter *recoding* di awal kata yang dipenggal. Karakter *recoding* adalah huruf kecil setelah tanda hubung (‘-’) dan terkadang berada sebelum tanda kurung. Sebagai contoh, kata “menangkap” (aturan 15), setelah dipenggal menjadi “nangkap”. Karena tidak *valid*, maka *recoding* dilakukan dan menghasilkan kata “tangkap”.
6. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai kata dasar. Proses selesai.

Contoh hasil proses *stemming* yang dilakukan dengan menggunakan *algoritma stemming* Nazief & Adriani dengan kata “mengecek” dilihatkan pada tabel 3.5

**Tabel 3.5** Contoh *stemming* kata “mengecek”

| Algoritma *stemming* Nazief & Adriani | Hasil Stemming | Keterangan |
|---|---|---|
| Cari kata dikamus kata dasar | mengecek | Tidak ditemukan kata pada kamus kata dasar |

| | | |
|---|---|---|
| Hilangkan partikel (“-lah”, “-kah”, “-tah” atau “-pun”) | mengecek | Tidak terdapat Partikel |
| Hilangkan kata ganti (“-ku”, “-mu”, atau “-nya”), | mengecek | Tidak terdapat Kata Ganti Kepunyaan |
| Hilangkan akhiran (“-i”, “-an” atau “-kan”) | mengecek | Tidak Terdapat akhiran |
| Hilangkan awalan (“di-”, “ke-”, “se-”, “me-”, “be”, “pe-”, “te-”) | ngecek | Hapus awalan “me-” |
| *recoding* | ecek | Pada aturan 17 tabel 2.3 dilakukan *recoding* pada kata mengecek, setelah dipenggal menjadi “ecek” dan dicek pada kamus tidak *valid* dan dilakukan lagi pengecekan dengan kata “kecek” juga tidak *valid* pada kamus kata dasar, sehingga menghasilkan kata “ecek” |
| Hasil *stemming* | ecek | |

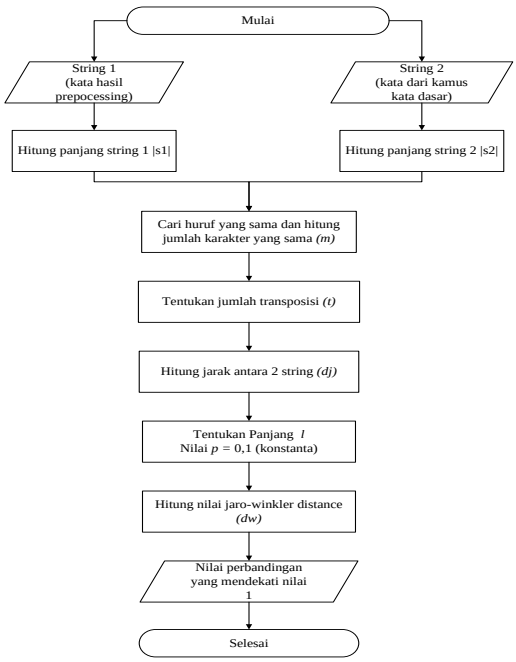
### 3.3.3 *Overstemming* atau *Understemming*

*Overstemming* dan *Understemming* adalah sebuah kesalahan dan kekurangan dari proses *stemming*. Kata yang mengalami *Overstemming*

atau *Understemming* tidak dapat ditemukan pada kamus kata dasar. Kata yang didapatkan bisa mengalami pemenggalan imbuhan yang lebih (*overstemming*) atau pemenggalan imbuhan yang terlalu sedikit (*understemming*). Seperti pada kata "mengecek" hasil dari *stemming*nya adalah "ecek". Jika kita lihat, kata "ecek" bukanlah sebuah kata dasar. Kata "ecek" mengalami *understemming* yaitu pemenggalan imbuhan yang terlalu sedikit, dimana pengguna bisa menganalisis kata dasar dari kata "ecek" adalah "cek" dari kata imbuhan "mengecek". Namun, apabila kata imbuhan yang melalui proses *stemming* ditemukan kata dasarnya maka, tidak diperlukan metode *jaro winkler* untuk mendapatkan kata dasar.

### 3.3.4 Hitung Jarak Kedekatan Kata Dengan Metode *Jaro Winkler*

Penelitian ini menggunakan metode *jaro winkler* dalam proses mengukur kedekatan antar kata. Kata yang sudah melalui proses *stemming* akan dilakukan perhitungan untuk menghetahui nilai kedekatan pada setiap kata dasar pada kamus sehingga bisa didapatkan kata dasar yang akurat. Alur proses dalam perhitungan kedekatan dengan *jaro winkler* seperti pada gambar 3.3




**Gambar 3. 3** Alur proses metode *Jaro Winkler*

Proses tahapan metode *jaro winkler* yang terjadi pada gambar 3.3:

1. Dimulai dengan 2 *string* sebagai inputan *string* 1 didapatkan dari hasil *prepocessing* dan *string* 2 didapatkan dari kamus kata dasar. Contoh kasus permasalahan dari hasil *stemming algoritma* Nazief & Adriani yaitu kata "ecek". Penganalisisan pengguna pada kata "ecek" adalah bukan sebuah kata dasar, melainkan kata yang mengalami *understemming* yaitu pemenggalan imbuhan yang terlalu sedikit. Dilihat dari kata imbuhan awal yaitu kata "mengecek", kata hasil *stemming* seharusnya adalah "cek". Karena kesalahan dari *algoritma* Nazief & Adriani kata tersebut menjadi "ecek". Maka dari itu "ecek" dianggap sebagai *string* 1 dan untuk *string* 2 adalah kata yang berada pada kamus kata dasar. Sebagai contoh kata yang diambil dari kamus kata dasar sebanyak 25 kata ditampilkan pada tabel 3.6 sebagai berikut:

   **Tabel 3.6** Tabel kamus kata dasar

| No | Kamus Kata |
|---|---|
| 1 | bebas |
| 2 | beda |
| 3 | belakang |
| 4 | benar |
| 5 | bencana |
| 6 | benci |
| 7 | bersih |
| 8 | cedera |
| 9 | cegah |
| 10 | cek |
| 11 | cekat |
| 12 | cekik |
| 13 | cela |

| No | Kamus Kata |
|---|---|
| 14 | celaka |
| 15 | cepat |
| 16 | kacau |
| 17 | kreatif |
| 18 | kecewa |
| 19 | kecil |
| 20 | kedip |
| 21 | kelompok |
| 22 | tangkap |
| 23 | tabrak |
| 24 | takdir |
| 25 | takut |
| | |

2. Setelah ditentukan *string* 1 dan *string* 2 proses selanjutnya yaitu menghitung panjang *string* atau huruf yang ada pada kata yang akan diproses. Pada proses ini juga sudah dimulai perbandingan antara *string* 1 dan *string* 2

   *String* 1 => ecek = 4

   *String* 2 => bebas = 5

   **Tabel 3.7** Perbandingan panjang *string* 1 dan *string* 2

| Perbandingan *String* 1 : *String* 2 | *String* 1 | *String* 2 |
|---|---|---|
| ecek : bebas | 4 | 5 |
| ecek : beda | 4 | 4 |
| ecek : belakang | 4 | 8 |
| ecek : benar | 4 | 5 |
| ecek : bencana | 4 | 7 |
| ecek : benci | 4 | 5 |
| ecek : bersih | 4 | 6 |
| ecek : cedera | 4 | 7 |
| ecek : cegah | 4 | 5 |
| ecek : cek | 4 | 3 |
| ecek : cekat | 4 | 5 |
| ecek : cekik | 4 | 5 |
| ecek : cela | 4 | 4 |
| ecek : celaka | 4 | 6 |
| ecek : cepat | 4 | 5 |
| ecek : kacau | 4 | 5 |
| ecek : kreatif | 4 | 8 |
| ecek : kecewa | 4 | 6 |
| ecek : kecil | 4 | 5 |
| ecek : kedip | 4 | 5 |

| | | |
|---|---|---|
| ecek : kelompok | 4 | 8 |
| ecek : tangkap | 4 | 7 |
| ecek : tabrak | 4 | 6 |
| ecek : takdir | 4 | 6 |
| ecek : takut | 4 | 5 |

3. Setelah panjang *string* 1 dan *string* 2 ditemukan, maka selanjutnya adalah mencari nilai *m* yaitu cari karakter yang sama dan hitung karakter yang sama.

   *string* 1 => ecek

   *string* 2 => bebas

   Pada perbandingan dua *string* tersebut diketahui masing-masing memiliki huruf E, maka nilai *m* = 1 karena terdapat 1 huruf yang sama.

   **Tabel 3.8** Jumlah Karakter yang sama (*m*)

| Perbandingan *string* 1 : *string* 2 | *m* |
|---|---|
| ecek : bebas | 1 |
| ecek : beda | 1 |
| ecek : belakang | 2 |
| ecek : benar | 1 |
| ecek : bencana | 2 |
| ecek : benci | 2 |
| ecek : bersih | 1 |
| ecek : cedera | 2 |
| ecek : cegah | 1 |
| ecek : cek | 3 |
| ecek : cekat | 3 |
| ecek : cekik | 3 |

| | |
|---|---|
| ecek : cela | 2 |
| ecek : celaka | 3 |
| ecek : cepat | 2 |
| ecek : kacau | 2 |
| ecek : kreatif | 2 |
| ecek : kecewa | 4 |
| ecek : kecil | 3 |
| ecek : kedip | 2 |
| ecek : kelompok | 2 |
| ecek : tangkap | 1 |
| ecek : tabrak | 1 |
| ecek : takdir | 1 |
| ecek : takut | 1 |

4. Proses selanjutnya yaitu menentukan *t* atau nilai *transpose* yaitu hitung karakter yang terdapat *transposisi* antara *string* 1 dan *string* 2.

   *String* 1 => ecek

   *String* 2 => bebas

   Kata ecek dan bebas tidak memiliki nilai *transposisi* karena dari struktur huruf dan jumlah huruf yang dimiliki masing-masing berbeda, maka t = 0.

   **Tabel 3.9** Jumlah *transposisi* (*t*)

| Perbandingan *String* 1 : *string* 2 | *t* |
|---|---|
| ecek : bebas | 0 |
| ecek : beda | 0 |
| ecek : belakang | 0 |
| ecek : benar | 0 |

| | |
|---|---|
| ecek : bencana | 0 |
| ecek : benci | 0 |
| ecek : bersih | 0 |
| ecek : cedera | 0 |
| ecek : cegah | 0 |
| ecek : cek | 0 |
| ecek : cekat | 0 |
| ecek : cekik | 0 |
| ecek : cela | 0 |
| ecek : celaka | 0 |
| ecek : cepat | 0 |
| ecek : kacau | 0 |
| ecek : kreatif | 0 |
| ecek : kecewa | 0 |
| ecek : kecil | 0 |
| ecek : kedip | 0 |
| ecek : kelompok | 0 |
| ecek : tangkap | 0 |
| ecek : tabrak | 0 |
| ecek : takdir | 0 |
| ecek : takut | 0 |

5. Setelah ditentukan panjang *string*, nilai *m*, dan nilai *t*. Maka dilanjutkan dengan menghitung nilai jarak kesamaan antara 2 string menggunakan rumus *dj*.

   ***Ecek : bebas***

   |**s**1| = 4

   |**s**2| = 5

m = 1

t = 0

$dj = \frac{1}{3} \times \left( \frac{m}{s1} + \frac{m}{s2} + \frac{m-t}{m} \right)$

$= \frac{1}{3} \times \left( \frac{1}{4} + \frac{1}{5} + \frac{1-0}{1} \right)$

$= \frac{1}{3} \times \left( \frac{5+4+20-0}{20} \right)$

$= \frac{1}{3} \times \left( \frac{29}{20} \right)$

$= \frac{29}{60}$

= 0,483

**Tabel 3.10** Jarak Kesamaan 2 *String* (*dj*)

| Perbandingan *String* 1 : *String* 2 | *dj* |
|---|---|
| ecek : bebas | 0,483333333 |
| ecek : beda | 0,5 |
| ecek : belakang | 0,583333333 |
| ecek : benar | 0,483333333 |
| ecek : bencana | 0,595238095 |
| ecek : benci | 0,633333333 |
| ecek : bersih | 0,472222222 |
| ecek : cedera | 0,595238095 |
| ecek : cegah | 0,483333333 |
| ecek : cek | 0,916666667 |
| ecek : cekat | 0,783333333 |
| ecek : cekik | 0,783333333 |
| ecek : cela | 0,666666667 |
| ecek : celaka | 0,75 |
| ecek : cepat | 0,633333333 |
| ecek : kacau | 0,633333333 |

| | |
|---|---|
| ecek : kreatif | 0,583333333 |
| ecek : kecewa | 0,888888889 |
| ecek : kecil | 0,783333333 |
| ecek : kedip | 0,633333333 |
| ecek : kelompok | 0,583333333 |
| ecek : tangkap | 0,464285714 |
| ecek : tabrak | 0,472222222 |
| ecek : takdir | 0,472222222 |
| ecek : takut | 0,483333333 |

6. Setelah didapatkan nilai *dj*, proses selanjutnya adalah mencari nilai *dw* yaitu nilai *jaro winklernya*, terlebih dahulu tentukan nilai *l* yaitu panjang karakter di awal *string* yang sama sampai ditemukan perbedaan, maksimalnya adalah 4. Nilai *p* bersifat konstanta yaitu 0,1.

   *String* 1 => ecek

   *String* 2 => bebas

   Pada *string* 1 dan 2 tidak terdapat kesamaan huruf pada awal kata yaitu huruf E dan B sehingga nilai *l* = 0.

   **Tabel 3.11** Panjang *l* dan Nilai *p*

| Perbandingan *String* 1 : *String* 2 | *l* | *p* |
|---|---|---|
| ecek : bebas | 0 | 0,1 |
| ecek : beda | 0 | 0,1 |
| ecek : belakang | 0 | 0,1 |
| ecek : benar | 0 | 0,1 |
| ecek : bencana | 0 | 0,1 |
| ecek : benci | 0 | 0,1 |
| ecek : bersih | 0 | 0,1 |

| | | |
|---|---|---|
| ecek : cedera | 0 | 0,1 |
| ecek : cegah | 0 | 0,1 |
| ecek : cek | 0 | 0,1 |
| ecek : cekat | 0 | 0,1 |
| ecek : cekik | 0 | 0,1 |
| ecek : cela | 0 | 0,1 |
| ecek : celaka | 0 | 0,1 |
| ecek : cepat | 0 | 0,1 |
| ecek : kacau | 0 | 0,1 |
| ecek : kreatif | 0 | 0,1 |
| ecek : kecewa | 0 | 0,1 |
| ecek : kecil | 0 | 0,1 |
| ecek : kedip | 0 | 0,1 |
| ecek : kelompok | 0 | 0,1 |
| ecek : tangkap | 0 | 0,1 |
| ecek : tabrak | 0 | 0,1 |
| ecek : takdir | 0 | 0,1 |
| ecek : takut | 0 | 0,1 |

7. Proses terakhir yaitu menghitung nilai *dw*, setelah ditentukan nilai *dj*, *l*, dan *p*. Hitung nilai *jaro winkler* (*dw*)

   ***Ecek : bebas***

   *dj* = 0,483

   *l* = 0

   *p* = 0,1

   *dw* = *dj + ( l x p (1 – dj ))*

   = 0,483 + ( 0 x 0,1 (1 – 0,483 ))

   = 0,483+ ( 0 ( 0,517))

   = 0,483+ (0)

= 0,483

**Tabel 3.12** Nilai *Jaro Winkler* (*dw*)

| Perbandingan *String* 1 : *String* 2 | *dw* |
|---|---|
| ecek : bebas | 0,483333333 |
| ecek : beda | 0,5 |
| ecek : belakang | 0,583333333 |
| ecek : benar | 0,483333333 |
| ecek : bencana | 0,595238095 |
| ecek : benci | 0,633333333 |
| ecek : bersih | 0,472222222 |
| ecek : cedera | 0,595238095 |
| ecek : cegah | 0,483333333 |
| ecek : cek | 0,916666667 |
| ecek : cekat | 0,783333333 |
| ecek : cekik | 0,783333333 |
| ecek : cela | 0,666666667 |
| ecek : celaka | 0,75 |
| ecek : cepat | 0,633333333 |
| ecek : kacau | 0,633333333 |
| ecek : kreatif | 0,583333333 |
| ecek : kecewa | 0,888888889 |
| ecek : kecil | 0,783333333 |
| ecek : kedip | 0,633333333 |
| ecek : kelompok | 0,583333333 |
| ecek : tangkap | 0,464285714 |
| ecek : tabrak | 0,472222222 |
| ecek : takdir | 0,472222222 |

| | |
|---|---|
| ecek : takut | 0,483333333 |

### 3.3.5 Pengurutan kata dasar dari nilai terbesar

Setelah proses mencari nilai *jaro winkler* setiap kata. Maka proses selanjutnya adalah mengurutkan nilai dari yang terbesar sampai yang terkecil untuk dicari kata yang paling mendekati nilai 1atau sama dengan 1.

**Tabel 3.13** Pengurutan kata dari yang nilai terbesar

| Perbandingan *String* 1 : *String* 2 | Pengurutan |
|---|---|
| ecek : cek | 0,916667 |
| ecek : kecewa | 0,888889 |
| ecek : cekat | 0,783333 |
| ecek : cekik | 0,783333 |
| ecek : kecil | 0,783333 |
| ecek : celaka | 0,75 |
| ecek : cela | 0,666667 |
| ecek : benci | 0,633333 |
| ecek : cepat | 0,633333 |
| ecek : kacau | 0,633333 |
| ecek : kedip | 0,633333 |
| ecek : bencana | 0,595238 |
| ecek : cedera | 0,595238 |
| ecek : belakang | 0,583333 |
| ecek : kreatif | 0,583333 |
| ecek : kelompok | 0,583333 |
| ecek : beda | 0,5 |
| ecek : bebas | 0,483333 |
| ecek : benar | 0,483333 |
| ecek : cegah | 0,483333 |
| ecek : takut | 0,483333 |
| ecek : bersih | 0,472222 |
| ecek : tabrak | 0,472222 |
| ecek : takdir | 0,472222 |

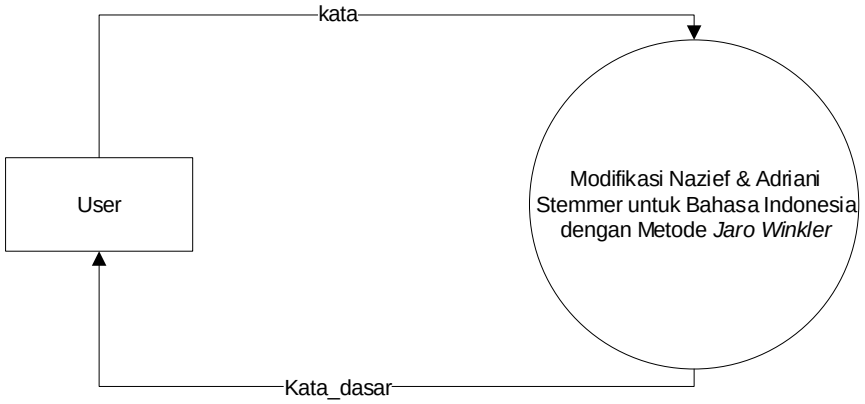
### 3.3.6 Kata dasar

Proses terakhir yaitu didapatkan kata dasar. Pada kata imbuhan "mengecek" yang diproses *stemming* menjadi "ecek" selanjutnya dibandingkan dengan kamus kata dasar dengan menggunakan metode *jaro winkler* didapatkan kata dasar yang tepat yaitu kata "cek", dapat dilihat pada tabel 3.13 pada pengurutan nilai *jaro winkler* kata "cek" mempunyai nilai paling besar yaitu 0,916666667.

## 3.4 Perancangan Sistem

### 3.4.1 Diagram Konteks

Diagram konteks merupakan diagram yang menunjukan sebuah proses tunggal dalam sistem yang berhubungan dengan bagian yang terkait. Rangkaian diagram konteks yang digunakan pada penelitian ini seperti pada gambar 3.4

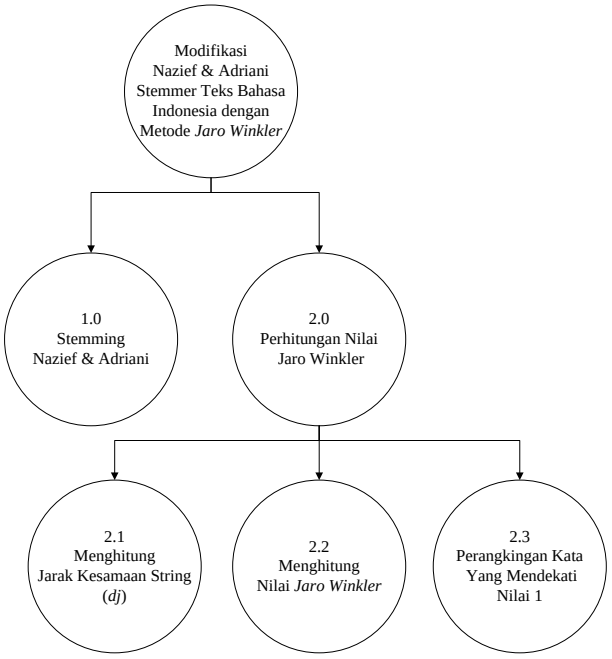



**Gambar 3.4** Diagram Konteks Modifikasi Nazief & Adriani Stemmer

Dari gambar 3.4 tersebut menggambarkan bahwa melibatkan satu pihak. *User* mengirimkan *input* berupa kata yang memiliki imbuhan seperti awalan (*prefix*), sisipan (*infix*), akhiran (*suffix*), dan gabungan awalan akhiran (*confixes*) yang digunakan sebagai data yang akan diproses. Setelah didapatkan hasil nilai kesamaan kata *(string)* maka *output* atau keluaran dari sistem berupa kata dasar yang paling mendekati.

### 3.4.2 Diagram Berjenjang

Diagram berjenjang sangat diperlukan dalam perancangan semua proses yang ada. Diagram berjenjang merupakan pengambaran proses dari awal sampai ke level-level berikutnya. Dalam penelitian Modifikasi Nazief & Adriani *Stemmer* ini mempunyai tiga level seperti pada gambar 3.5




**Gambar 3.5** Diagram berjenjang di Modifikasi Nazief & Adriani *Stemmer*

Berikut penjelasan gambar 3.5 berdasarkan kerangka diagram berjenjang diatas terlihat bahwa sistem yang dibuat terdiri dari dua level

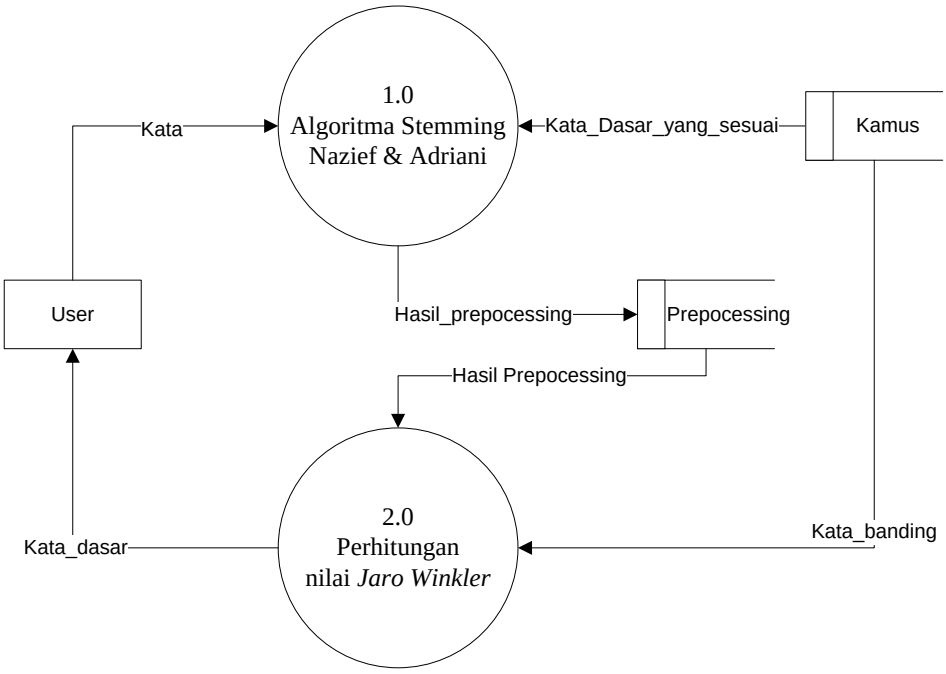
1. Top level : Modifikasi Nazief & Adriani Stemmer Teks Bahasa Indonesia Dengan Metode *Jaro Winkler*
2. Level 0 : merupakan hasil *break down* dari proses keseluruhan dari Modifikasi Nazief & Adriani Stemmer Teks Bahasa Indonesia Dengan Metode *Jaro Winkler* menjadi beberapa sub proses yaitu:
    a. *Stemming* Nazief & Adriani
    b. Perhitungan nilai *Jaro Winkler*
3. Level 1 : merupakan sub proses dari beberapa proses pada level 0 dalam Modifikasi Nazief & Adriani *Stemmer* Teks Bahasa Indonesia

Dengan Metode *Jaro Winkler* yang menggambarkan beberapa proses detail yaitu :

1. Hasil dari sub proses Perhitungan nilai *Jaro Winkler* :
    a. Menghitung jarak kesamaan *string* (*dj*)
    b. Menghitung nilai *Jaro Winkler* (*dw*)
    c. Perangkingan kata yang mendekati nilai 1

### 3.4.3 *Data Flow Diagram*

#### 3.4.3.1 *Data Flow Diagram Level 0*



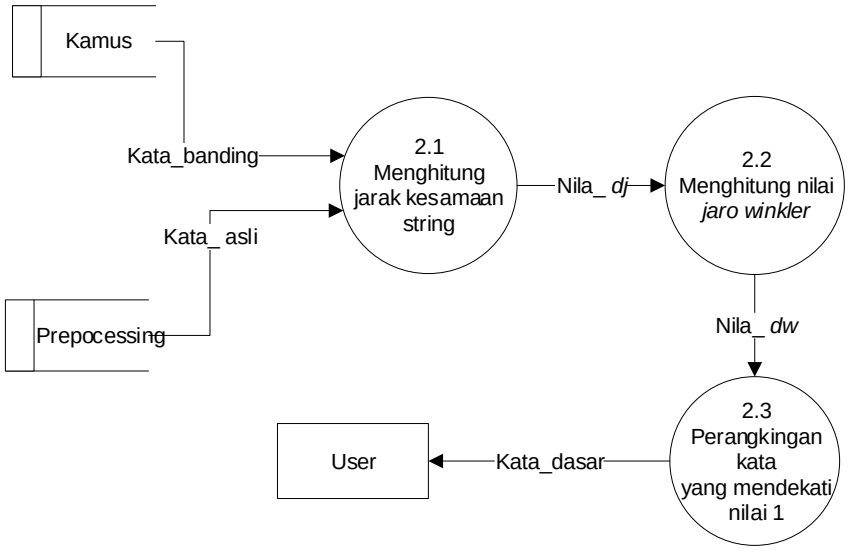

**Gambar 3.6** *Data Flow Diagram* level 0 Modifikasi Nazief & Adriani *Stemmer*

Berdasarkan pada gambar 3.6, dapat dijelaskan bahwa DFD level 0 menjelaskan beberapa proses yang terjadi. Beberapa proses yang ada pada DFD level 0 antara lain :

*a. Prepocessing*
b. Perhitungan nilai *Jaro Winkler*

### 3.4.3.2 *Data Flow Diagram Level 1*

#### 1) DFD Level 1 Proses 2




**Gambar 3.7** *Data Flow Diagram* level 1 Proses 2 Modifikasi Nazief & Adriani *Stemmer*

Berdasarkan pada gambar 3.7 dapat dijelaskan bahwa DFD Level 1 proses 2 menjelasakan tiga proses detail dari proses perhitungan *jaro winkler* yakni :

a. Menghitung jarak kesamaan *string*
b. Menghitung nilai *jaro wiinkler*
c. Perangkingan kata yang mendekati nilai 1

## 3.5 Perancangan Basis Data

Basis data adalah kumpulan file-file yang mempunyai kaitan antara satu file dengan file lain sehingga membentuk suatu bangunan data untuk menginformasikan suatu hasil dari sebuah proses. Berikut untuk struktur dan desain tabel dari *database* yang digunakan dalam proses pembuatan sistem modifikasi Nazief & Adriani *Stemmer* berbahasa indonesia .

### 3.5.1 Desain Tabel

Desain tabel merupakan susunan dari tabel yang akan digunakan atau diimplementasikan kedalam *database*, dimana desain tabel memuat detail tipe data dan *primary key* serta *foreign key* dari tabel tersebut.

1. Tabel Kamus Kata Dasar

   Tabel Kata dasar berisi kumpulan kata dasar dari sebuah kata berbahasa indonesia yang nantinya akan digunakan pada proses *stemming* dan juga sebagai kata sumber yang nanti akan menjadi pembanding untuk kata pembanding pada proses metode *jaro winkler*. Struktur tabel kamus kata dasar seperti pada tabel 3.14

   **Tabel 3.14** Tabel Kata Dasar

| Nama Kolom | Tipe | Ukuran | Keterangan |
|---|---|---|---|
| Id_katadasar | int | 15 | *Primary key* |
| Kata_dasar | varchar | 50 | |
| Jumlah_Huruf_KD | int | 10 | |

2. Tabel kata

   Tabel kata merupakan tempat untuk menyimpan kata sebelum dilakukan proses *prepocessing*. Struktur tabel kata seperti pada tabel 3.15

   **Tabel 3.15** Tabel Kata

| Nama Kolom | Tipe | Ukuran | Keterangan |
|---|---|---|---|
| Id_kata | int | 15 | *Primary key* |
| Kata | varchar | 50 | |

3. Tabel *prepocessing*

   Tabel *prepocessing* merupakan tempat kata yang telah melalui tahapan proses *prepocessing* dan menjadi kata pembanding untuk dicari nilai *jaro winkler*. Struktur tabel *preppocessing* seperti pada tabel 3.16

**Tabel 3.16** Tabel *Prepocessing*

| Nama Kolom | Tipe | Ukuran | Keterangan |
|---|---|---|---|
| Id_prepocessing | int | 15 | *Primary key* |
| Id_kata | int | 15 | *Foreign key* |
| Case_folding | varchar | 50 | |
| Jumlah_huruf_kata | int | 10 | |

4. Tabel *Histori*

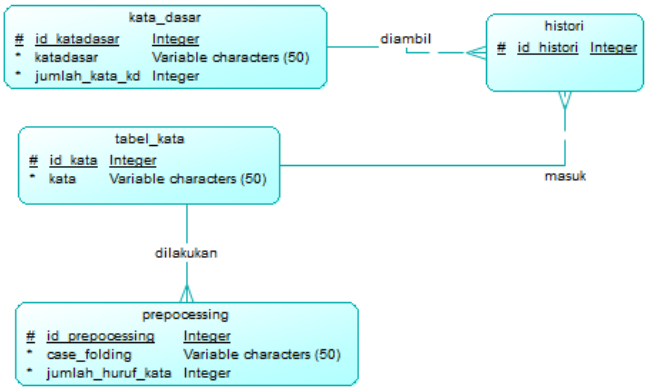
Tabel *histori* merupakan tempat penyimpanan kata yang telah melalui tahapan metode *jaro winkler*. Struktur tabel *histori* seperti pada tabel 3.17

**Tabel 3.17** Tabel Histori

| Nama Kolom | Tipe | Ukuran | Keterangan |
|---|---|---|---|
| Id_histori | int | 15 | *Primary key* |
| Id_prepocessing | varchar | 50 | *Foreign key* |
| Id_katadasar | int | 10 | |

### 3.5.2 *Entity Relationship Diagram*

*Entity relationship diagram* (ERD) merupakan model konseptual yang mengambarkan hubungan antar tabel yang ada. ERD digunakan untuk memodelkan struktur data dan hubungan antar data. Desain *entity relationship diagram* pada pembuatan implementasi modifikasi Nazief & Adriani seperti pada gambar 3.8




**Gambar 3.8** *Entity relationship diagram* dalam Modifikasi Nazief & Adriani *Stemmer*

## 3.6 Perancangan Antar Muka

Rancangan antarmuka (*interface*) berfungsi sebagai alat komunikasi antara sistem dengan pengguna. Antarmuka akan memberikan informasi berupa tampilan disertai dengan data-data yang diminta oleh pengguna. Dalam penelitian ini desain antarmuka dapat digunakan sebagai media pemasukan kata yang akan diproses dan menampilkan hasil kata yang telah diproses.

### 3.6.1 Halaman Beranda

Halaman beranda merupakan halaman pertama yang muncul ketika *user* membuka sistem. Halaman ini berisikan informasi mengenai nama sistem dan kegunaannya. Tampilan rancangan halaman beranda seperti pada gambar 3.9


**Gambar 3.9** Tampilan rancangan halaman beranda

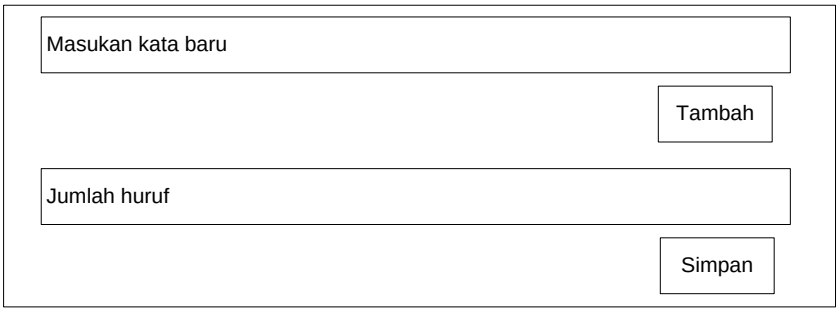
### 3.6.2 Halaman Kamus

Halaman kamus merupakan halaman yang berfungsi untuk menampilkan kumpulan kamus kata dasar. Pada halaman ini terdapat tombol tambah yang berfungsi untuk menambahkan kata dasar. Pada kolom aksi terdapat tombol hapus yang berfungsi untuk menghapus setiap kata. Tampilan rancangan halaman kamus seperti pada gambar 3.10


**Gambar 3.10** Tampilan rancangan halaman kamus

### 3.6.3 Halaman Tambah Kamus

Halaman tambah kamus merupakan halaman yang berfungsi untuk menambahkan kata baru pada kamus. Untuk mengakses halaman ini, pengguna harus menekan tombol tambah pada halaman kamus. Tampilan rancangan halaman tambah kamus seperti pada gambar 3.11



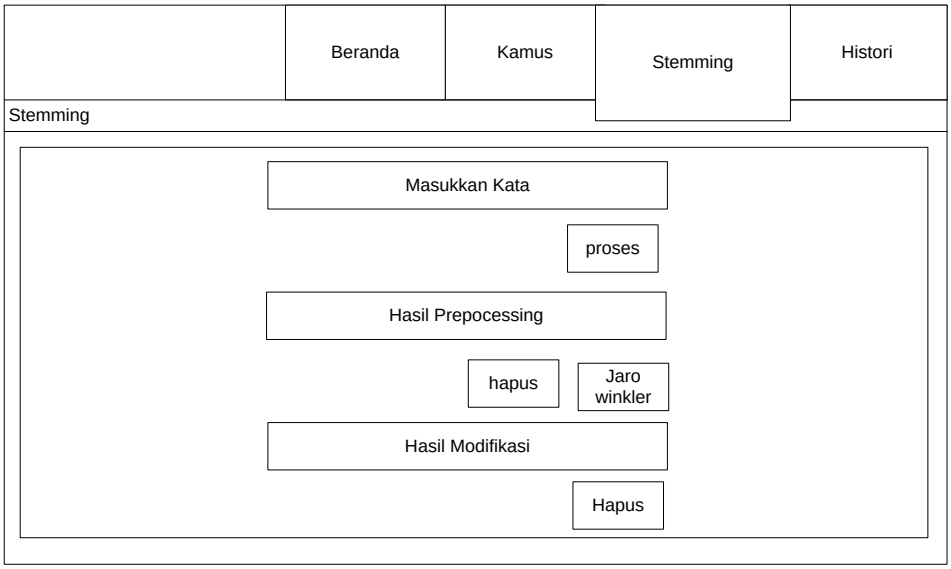

**Gambar 3.11** Tampilan rancangan halaman Tambah kamus

Pengguna dapat menginputkan sebuah kata dasar pada kolom masukan kata baru dan dilanjutkan dengan menekan tombol tambah, maka pada kolom jumlah huruf akan secara otomatis menghitung jumlah huruf yang diinputkan pada kolom masukan kata baru. Selanjutnya tekan tombol simpan untuk menyimpan kata pada kamus.

### 3.6.4 Halaman *stemming*

Halaman *stemming* merupakan halaman yang berfungsi untuk menginputkan kata yang akan dilakukan proses *stemming* dan pengukuran *jaro winkler*. Pada kolom masukkan kata berfungsi untuk tempat penginputan kata berimbuhan. Jika pengguna menekan tombol proses maka hasil *output* akan terlihat pada kolom hasil *prepocessing*. Jika *output* yang ditampilkan pada kolom hasil *prepocessing* belum mendapatkan seperti yang diinginkan, pengguna bisa menginputkan *output* tersebut untuk dicari kesamaannya pada kamus dengan menekan tombol *jaro winkler*, maka pada kolom hasil modifikasi akan menampilkan *output* dari hasil *output* kolom hasil *prepocessing*, dan apabila kata yang ditampilkan adalah kata yang diinginkan dan proses dianggap selesai pengguna bisa menghapus inputan dengan menekan tombol hapus yang terletak disamping tombol *jaro winkler*. Jika proses selesai sampai kolom hasil modifikasi dan ingin mengulangi kata yang

lain kembali maka bisa menekan tombol hapus yang ada dibawah kolom hasil modifikasi. Tampilan rancangan halaman *stemming* seperti pada gambar 3.12




**Gambar 3. 12** Tampilan rancangan halaman *stemming*

### 3.6.5 Halaman *Histori*

Halaman *histori* merupakan halaman yang berfungsi untuk melihat *Histori* pencarian kata dasar. Halaman ini berisi kata dasar yang pernah dicari. Pada halaman ini akan ditampilkan tabel yang berisi kata imbuhan, kata dasar, dan aksi. Kolom kata imbuhan berisi kata yang telah diinputkan pada kolom masukan kata pada halaman *stemming*. Kolom kata dasar berisi *output* dari kolom hasil *prepocessing* atau hasil modifikasi dimana apabila pengguna terakhir menekan tombol proses pada halaman *stemming* maka outputnya yaitu kata yang berada pada kolom hasil *prepocessing* tapi apabila pengguna terakhir menekan tombol *jaro winkler* pada halaman *stemming*. Maka *outputnya* sama dengan *ouput* yang dihasilkan pada kolom hasil modifikasi. Pada kolom kata aksi berfungsi untuk menghapus setiap baris pada kolom *histori*. Ada pula tombol hapus semua yang berfungsi menghapus semua *histori*. Tampilan rancangan halaman *histori* seperti pada gambar 3.13


**Gambar 3. 13** Tampilan rancangan halaman *histori*

## 3.7 Skenario Pengujian dan Evaluasi Sistem

Pada penelitian ini, untuk mengukur evaluasi kinerja sistem temu kembali informasi digunakan pengujian akurasi.

**Tabel 3.18** Parameter menghitung akurasi

| **Keterangan** | **Relevan** | **Tidak Relevan** |
|---|---|---|
| Terambil | True (T) | False (F) |

Rumus untuk menghitung Akurasi:

$$Akurasi = \frac{RW}{W} \; x \; 100 \; \% \qquad (3.1)$$

Nilai akurasi dinyatakan dalam persen. Semakin tinggi nilai tersebut menunjukkan semakin baiknya kinerja aplikasi. Evaluasi yang akan dilakukan dalam penelitian ini adalah menghitung nilai dari akurasi berdasarkan kata dasar yang ditemukan. Sedangkan untuk menentukan nilai dari akurasi harus didapatkan jumlah kata yang relevan terhadap suatu kata dasar pada kamus.

## 3.8 Spesifikasi Pembuatan Sistem

Kebutuhan perangkat lunak serta perangkat keras dari sistem sebagai berikut :

a. Kebutuhan Perangkat Lunak
   1. *Windows 10* sebagai sistem operasi yang digunakan.
   2. PHP dan *Sublime* sebagai bahasa pemrograman berbasis desktop dan sekaligus *compiler*nya.
   3. *SQLyog* sebagai database server.
   4. *XAMPP Control Panel*

b. Kebutuhan Perangkat Keras
   1. Komputer Intel pentium 2,0 GHz sekelas atau lebih tinggi
   2. RAM 2 GB atau lebih
   3. Hardisk dengan kapasitas 500 gigabyte atau lebih
   4. Monitor, mouse, keyboard standard