 ISSN:2302-8149
e-ISSN:2540-9719

Implementasi Algoritma *Levenshtein Distance* dan Metode *Regular Search Expression* dalam Mendeteksi Salah Ketik pada Javascript

Implementation of the Levenshtein Distance Algorithm and the Regular Search Expression Method for Detecting Typors in Javascript

[1]Mu'alif Lihawa, [2]Anggit Dwi Hartanto, [3]Norhikmah*, [4]Donni Prabowo, [5] Ika Nur Fajri, [6]Wiwi Widayani
[1,2,3,4,5,6]Program Studi Sistem Informasi, Fakultas Ilmu Komputer, Universitas Amikom Yogyakarta, Jalan Ringroad Utara, Condongcatur, Depok, Sleman, Yogyakarta Indonesia 55283
*e-mail: hikmah@amikom.ac.id

(***received*:** 20 Maret 2023, ***revised*:** 7 April 2023, ***accepted*:** 29 April 2023)

Abstrak

Mengetik adalah suatu kegiatan untuk menuliskan sebuah tulisan dalam bentuk cetakkan yang sudah dirangkai oleh mesin ketik. Dengan perkembangan zaman yang cepat mesin ketik digantikan oleh komputer karena efisien dalam membuat sebuah tulisan atau teks. sebuah teks atau tulisan yang mudah dipahami dalam penyampaian informasi tidak memiliki kesalahan kata yang mengakibatkan ketidakjelasan dalam informasi yang disampaikan. Dalam aplikasi pengolah kata seperti microsoft office word, memiliki fitur *suggestions word* dan *Autocorrect word* yang dimana sangat berguna dalam mengecek sebuah tulisan yang terdapat kesalahan kata dalam penulisannya. Penelitian ini mengembangkan sebuah *library* javascript untuk mendeteksi kesalahan *typo* terhadap penulisan kata yang salah dan merekomendasikan kata yang tepat untuk merubah kata yang salah. Penelitian ini menggunakan Algoritma Levenshtein Distance dan metode *Regular Search Expression.* Hasil dari penelitian ini berhasil diterapkan pada fitur rekomendasi kata pada library dengan nilai akurasi yang diperoleh mencapai 50% dan tingkat presisi 5%.

Kata kunci: *Regular Search Expression, Regex, Library, Javascript, Saltikjs, Levenshtein Distance*.

Abstract

Typing is an activity to write an article in printed form that has been assembled by a typewriter. With the rapid development of the times, typewriters were replaced by computers because they were efficient in making writing or text. a text or writing that is easy to understand in conveying information does not have word mistakes that result in unclear information being conveyed. In word processing applications such as Microsoft Office Word, it has the word suggestions and autocorrect word features which are very useful in checking an article where there are word errors in the writing. This research develops a javascript library to detect typo errors for writing wrong words and recommends the right words to change the wrong words. This study uses the Levenshtein Distance Algorithm and the Regular Search Expression method. The results of this study were successfully applied to the word recommendation feature in the library with an accuracy value of 50% and a precision level of 5%.

Keywords: *Regular Search Expression, Regex, Library, Javascript, Saltikjs, Levenshtein Distance*.

 ISSN:2302-8149
e-ISSN:2540-9719

1 Pendahuluan

Mengetik adalah suatu kegiatan untuk menuliskan sebuah tulisan dalam bentuk cetakan yang sudah dirangkai oleh mesin ketik. Mesin ketik mesin ketik digitalisasikan oleh komputer karena lebih efisien dalam membuat sebuah tulisan atau teks. Didalam komputer memiliki begitu banyak aplikasi pengolah kata seperti microsoft office word, microsoft office word memiliki fitur *suggestions word* dan *autocorrect word* yang dimana fungsi ini mendeteksi kata dan merekomendasikan kata yang salah ketik. Fitur tersebut jarang dijumpai di text editor pada website, maka dengan begitu dalam penelitian ini penulis akan membuat *library* javascript bernama saltikjs yang akan menerapkan fitur yang mirip dengan *suggestions word* dan *autocorrect word* dengan menggunakan menerapkan algoritma l*evenshtein distance* sebagai algoritma pencocokan antara 2 *string* dan metode r*egular search expression* sebagai metode pencarian string pada text editor.

Metode *Regular Search Expression* banyak digunakan dalam *search engine* favorit yang sering kita gunakan seperti Google, Yahoo, dst. dan penggunaan dari metode ini juga.Sangat luas dan tidak banyak digunakan dalam search engine, tetapi metode ini bisa ditemukan hampir dalam setiap fitur atau aplikasi yang memiliki fitur "*find*" atau "*replace*". Algoritma *Levenshtein Distance* adalah suatu pengukuran untuk menghitung jumlah perbedaan antara dua kata. Perhitungan jarak antara dua kata ditentukan dari jumlah minimum operasi perubahan untuk mengubah kata A menjadi kata B .*Algoritma Binary search* memiliki kelebihan dalam melakukan pencarian pada data berjumlah besar dengan keadaan terurut serta memiliki iterasi yang lebih efektif. Sedangkan *Regular Expression Search* memiliki kelebihan dalam melakukan pencarian yang tidak diketahui secara lengkap mengenai hasil dan kunci, selain itu algoritma ini juga memungkinkan untuk melakukan pencarian berdasarkan pola tertentu pada data[1].

Teknik *Regular Expressions* digunakan untuk mengetahui pola perilaku pengunjung website yang berfokus untuk mendeteksi adanya serangan pada website sehingga para administrator website dapat melakukan pencegahan maupun proses perbaikan pada website yang dikelola.. Dengan teknik ini ditemukan aktivitas berbahaya berupa 46 serangan *Cross Site Scripting* dan 983 serangan *Path Traversal*, total serangan sebanyak 1029 serangan yang merupakan aktivitas berbahaya pada website. [2]. Membangun *library* javascript yang berfungsi untuk mendeteksi salah ketik atau kata yang *typo*. Tujuan dalam penelitian ini adalah membuat fitur rekomendasi kata pada saltikjs dan mengimplementasikan algoritma *Levenshtein Distance* untuk mendeteksi kesamaan kata. Serta mengimplementasikan *Regular Search Expression* dalam pencarian kata pada *library* saltikjs.

2 Tinjauan Literatur

Algoritma Binary Search dan algoritma *Regular Search Expression*(REGEX) akukan untuk menyelesaikan permasalahan dalam sistem pencarian, sehingga algoritma pencarian dapat diterapkan lebih tepat dan efektif lagi[1]. Metode RegEx (*Regular Expression*) yang merupakan suatu notasi fleksibel dan ringkas untuk mencari dan menggantikan pola teks[3].*Binary Search* dan *Regular Search Expression* yang mana lebih baik dalam menemukan *string*. Dari pengujian tersebut membuktikan bahwa algoritma *Binary Search* tidak dapat menemukan *string* apabila kunci tidak tepat sesuai dengan hasil yang diharapkan. Sebaliknya, pada algoritma *Regular Search Expression* memungkinkan kita untuk tetap menemukan *string* walaupun kunci yang dimasukkan tidak tepat sesuai dengan hasil yang diharapkan. Kesamaan penelitian penulis buat adalah sama sama mengimplementasikan *Regular Search Expression* dengan javascript [1]. Menganalisis *Log Web Server* Menggunakan Teknik *Regular Expressions*. Kelebihan dari sistem ini, dalam menganalisis semua website yang bisa diakses dan kekurangan sistem ini adalah saat sistem sedang melakukan proses memecah data akses *log* pada basis 8data, sistem cenderung lama dalam pemrosesan. Kesamaan penelitian ini dengan penulis buat adalah sama sama menggunakan *regular expressions* sebagai metode atau algoritma dalam pembuatan aplikasi tetapi hanya beda dalam cara implementasi dari *regular expressions* [2].

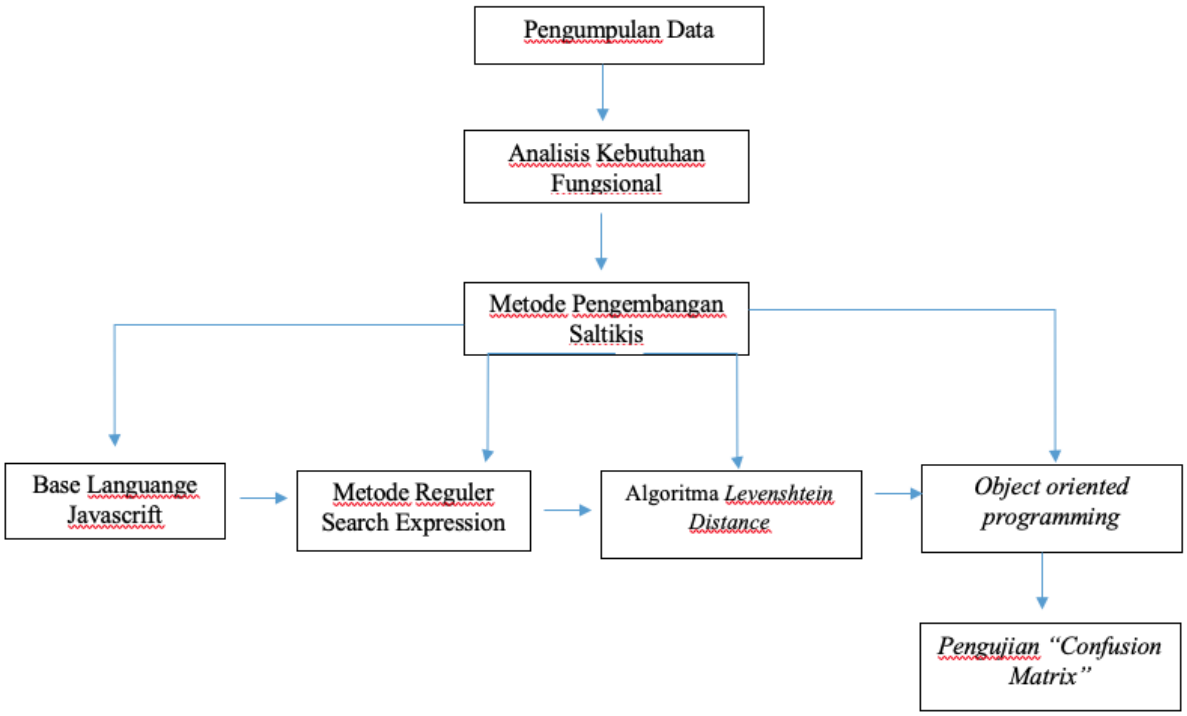
Menggunakan algoritma *Levenshtein Distance* untuk mencari kesamaan kata dan validasi kata, pencarian kata menggunakan metode *approximate string matching*. Metode *regular search expression*

 ISSN:2302-8149
e-ISSN:2540-9719

sebagai metode pencarian kata [4]. Algoritma Levenshtein Distance mengungguli algoritma adaptif. Proses preprocessing yang terdiri dari metode *case folding*, *tokenizing*, *stopword removal*, dan *stemming* yang dapat melakukan estimasi proses sistem menjadi lebih cepat. Algoritma *Levenshtein Distence* dapat mendeteksi plagiasi dengan baik dan rata-rata lama proses sistem tanpa dilakukan preprocessing adalah 6,283 ms dan dengan preprocessing adalah 4,920 ms.[5]. Algoritma Levenshtein distance merupakan algoritma yang cocok untuk diterapkan dalam mendeteksi *plagiarisme* karena memberikan nilai kemiripan dari dua string yang sejenis[6], [7], *Levenshtein Distance* bisa digunakan Sebagai *Spell Checker* Berbasis Android [[8] Algoritma *Levenshtein Distance* untuk mengoptimalisasi pencarian agar waktu pencarian menjadi lebih efisien dan menambahkan fitur *autocomplete* yang berfungsi untuk memberikan prediksi atau saran kata sesuai dengan karakter yang diketikkan oleh pengguna sehingga membantu pengguna agar lebih mencari kata yang ingin dicari [9]. Levenshtein Distance dimana algoritma ini digunakan untuk menentukan jarak levenshtein berdasarkan nilai yang paling terkecil atau paling sedikit jumlah modifikasinya[10]. Metode *Levenshtein Distance* digunakan untuk mendeteksi banyaknya kandidat katasesuai dengan typographical erroryang sudah teridentifikasi[11]. Proses pendeteksian diawali dengan melakukan tahap *preprocessing*,setelah itu masuk ketahap pencocokan *string* menggunakan matrik *edit distance*[12]**.** Pada penelitian ini menggunakan beberapa algoritma levenshtein distance dan metode regular search expression dalam mendeteksi salah ketik pada javascript dengan menggunakan menggunakan *object oriented programming* pada saltikjs, dengan metode pengujian menggunakan confusion matrix

3 Metode Penelitian

Metode Penelitian yang dilakukan dalam penelitian ini adalah sebagai berikut :



Gambar 1. Alur Penelitian

Pada Gambar 1 dijelaskan pada tahapan pertama teknik pengumpulan data yaitu studi literatur terkait metode *Regular Search Expression*, algoritma *Levenshtein Distance* dan metode *object oriented programming* pada javascript, tahapan kedua Analisis kebutuhan fungsional yang digunakan dalam library saltikjs,taha ketiga metode pengembangan saltikjs**,** a) Javascript sebagai *base language* dari *library* saltikjs untuk membuat fungsi mendeteksi salah kata yang dimasukkan oleh user. Javascript

 ISSN:2302-8149
e-ISSN:2540-9719

salah satu bahasa pemrograman yang berjalan disisi client melalui web browser yang disisipkan pada dokumen HTML. Javascript berorientasi objek dan bersifat *loosely typed*, yang artinya tidak perlu menentukan jenis informasi apa yang akan disimpan dalam variabel terlebih dahulu. Penulis menambahkan suatu hal tentang perkembangan javascript bahwa dengan adanya nodejs yaitu platform yang dibangun menggunakan javascript sebagai sintaksnya[13].b) Algoritma *Levenshtein Distance* ke dalam *library* saltikjs, digunakan untuk mencari kesamaan kata yang salah dengan kata yang ada pada kamus. *Levenshtein Distance* merupakan algoritma pengukuran kesamaan kata A dan Kata B yang menghitung jarak antara dua kata tersebut, algoritma ini menghitung jarak antara dua kata ditentukan dari jumlah minimum operasi perubahan untuk mengubah kata A menjadi kata B[2] Terdapat tiga macam operasi yang dapat dilakukan oleh algoritma ini yaitu, operasi melibatkan penyisipan (*insertion*), penghapusan (*deletion*), dan penggantian (*substitution*) dari suatu karakter tunggal [13]

$$similarity = \left(1 - \frac{d(m,n)}{max(S,T)}\right) * 100\%. \quad (1)$$

Pada persamaan (1) adalah rumus menghitung *similarity* dengan persamaan, yaitu dengan D[m, n] adalah nilai jarak, terletak pada baris ke m dan kolom ke n, S adalah panjang string awal, T adalah panjang string target, dan *Max*(S, T) adalah panjang string terbesar antara string awal dan *string* akhir. Bobot *similarity* diasumsikan pada rentang 0 hingga 100 persen, yang artinya nilai 100 adalah nilai maksimum yang menunjukkan bahwa kedua kata tersebut sama identik [13]. c) Metode *Regular Search Expression* ke dalam *library* saltikjs, digunakan sebagai metode pencarian *string* di dalam *library* ini, Metode *Regular Search Expression* bisa juga disebut dengan RegExp atau Regex Atau RE, Adalah metode pencarian yang mencari sebuah string menggunakan *Regular Expression*. *Regular Expression* itu sendiri adalah adalah serangkaian karakter yang mendefinisikan sebuah pola pencarian. Yang membedakan Regex dari string biasa adalah terdapatnya karakter-karakter khusus yang disebut karakter meta (*metacharacters*). Karakter-karakter ini tidak akan dicocokkan secara literal dengan karakter itu sendiri, tapi mewakili sekelompok karakter lain atau pola khusus tertentu [1].d) *Object oriented programming* pada saltikjs, agar mudah dalam pemanggilan suatu fungsi dalam *library* . Pemrograman berorientasi objek adalah suatu cara penulisan dalam membuat program yang dimana penulisannya memiliki struktur data dan fungsi tertentu. objek adalah suatu entitas yang memiliki fungsi tersendiri, setiap objek bisa memiliki hubungan antara sesama objek [16].tahapan keempat yaitu Pengujian menggunakan algoritma menggunakan confusion matrix dan pengujian sistem menggunakan blackbox

4 Hasil dan Pembahasan

Berikut hasil dan pembahasan penelitian implementasi algoritma *Levenshtein Distance* dan metode *Regular Search Expression* dalam mendeteksi salah ketik Pada Javascript

4.1 Analisis Kebutuhan

Analisis kebutuhan fungsional dalam *library* saltikjs adalah 1).Menampilkan *mark* terhadap kata yang tidak ada pada kamus yang ditentukan.2). Menu rekomendasi kata pada posisi kata yang di *mark*. 3). Dapat mengganti kata yang salah dengan kata yang benar.4). Membersihkan seluruh *mark* yang telah disematkan.

4.2 Metode Pengembangan

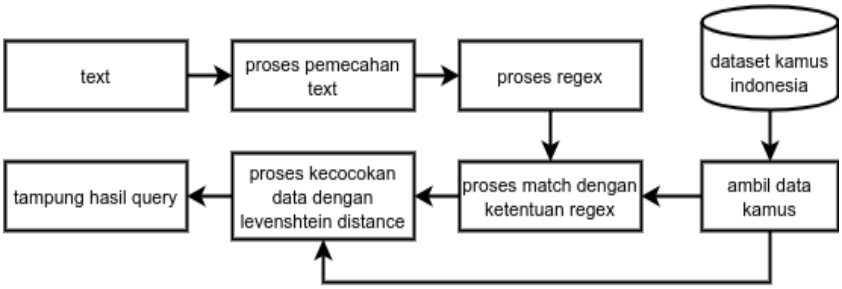
a. *Base Language* Saltikjs

 ISSN:2302-8149
e-ISSN:2540-9719

Library ini memiliki fitur mendeteksi kata yang salah dan menggantinya dengan kata yang benar, dan saltikjs ini akan tulis dengan javascript sebagai *base language* nya.

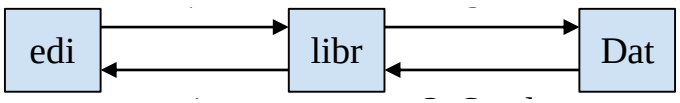
b. Perancangan Sistem Dengan Algoritma Levenshtein Distance Dan Metode Regular Search Expression

Sebelum dilakukan implementasi algoritma dan metode yang digunakan ke dalam sistem, maka akan dilakukan pemodelan atau desain sistem dalam implementasi algoritma *levenshtein distance* dan metode *regular search expression*. Dalam gambaran umum desain sistem untuk implementasi algoritma dan metode yang digunakan seperti Gambar 2.



Gambar 2. Desain Sistem Penggunaan Algoritma dan Metode yang digunakan

Proses pertama yang dilakukan adalah memasukkan *text* dan memecahkan *text* tersebut menjadi kata sehingga akan terbentuk beberapa kata dari *text* yang telah dipecahkan. Hasil dari pemecahan tersebut akan diproses dengan regex untuk mengubah kata menjadi *pattern* yang diinginkan. Setelah membuat kata berubah menjadi *pattern*, *pattern* tersebut akan digunakan pada proses *matching* kata yang sudah mengikuti ketentuan regex, dan akan dilanjutkan dengan proses pencocokan kata dengan kata pada kamus. Hasil dari pencocokan tersebut akan dicocokan kemiripannya dengan algoritma *levenshtein distance* untuk mencari kata yang mirip, lalu hasil dari proses tersebut akan ditampung ke dalam data *array* yang akan ditampilkan ke dalam rekomendasi kata.

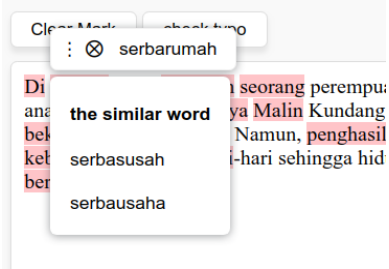


Gambar 3. Alur Kerja Sistem

Pada Gambar 3 dijelaskan alur kerja dari sistem pada *library*, dimana editor yang berisikan teks akan diambil oleh *library* dan diproses menjadi sebuah data *array* setelah itu diproses lagi untuk mencari kata yang salah ketik dengan fungsi pencarian yang sudah dibuat di *library*. Setelah melakukan tahapan proses pada *library*, ada suatu fungsi untuk *request* data kamus.

Proses mencari kesamaan kata ini membantu untuk mencari kata yang sama dengan kata yang salah. Misalnya jika mempunyai pola kata “sebuah” maka fungsi kata yang serupa akan mencari kata yang serupa dengan pola kata tersebut.

 ISSN:2302-8149
e-ISSN:2540-9719



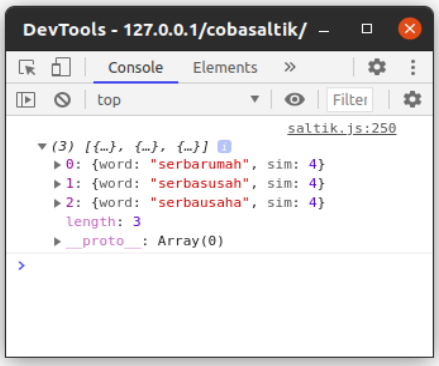
Gambar 4. Ouput Kata Yang Serupa Dengan Pola "sebuah"

Pada gambar 4 diperlihatkan output dari hasil proses pencarian kata yang serupa dengan pola "sebuah". Pada fungsi pencarian kata yang serupa memiliki kondisi dimana terdapat regex yang sudah dibuat dengan pola "sebuah" dan fungsi *levenshtein distance*. Fungsi dari *Levenshtein Distance* mencari kata yang mendekati berdasarkan urutan karakter yang dicocokan.

Ada data kamus seperti di Tabel 1**.** Akan dibandingkan dengan pola kata "sebuah" dari data kamus tersebut.

Tabel 1. Data Kamus

Kamus dummy bahasa indonesia				
aba	abad	...	serbaruma h	serbasusah
serbausaha	serbet	...	tragis	trafo
trama	...	zohal	zona	...
Jumlah data ada 29931 kata				



Gambar 5. Output Fungsi *Levenshtein Distance*

Pada Gambar 5 diperlihatkan *output* dari fungsi *levenshtein distance*, terlihat object sim yang mengeluarkan angka untuk nilai kemiripan antara kata yang dibandingkan dengan kata "sebuah", semakin kecil angka yang dikeluarkan maka kata tersebut serupa dengan kata yang dibandingkan. Dari pola "sebuah" yang dibandingkan dengan data pada Tabel 1. Terlihat jelas kata diperlihatkan berjumlah 3 kata, dikarenakan sebelum menggunakan fungsi *levenshtein distance* proses regex yang digunakan untuk menyaring data yang sesuai panjang karakter kata dan akan ditambahkan 4 untuk panjangnya dan *matching* pada kata "sebuah", diimplementasi javascript tidak menggunakan satuan % karena lebih lebih menghemat baris kode pada *library*.

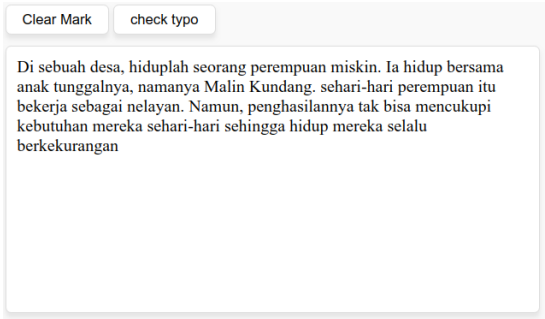
c. *Object oriented programming* pada saltikjs.

Berikut hasil tampilan antar muka implementasi Object oriented programming pada saltikjs:

1. Tampilan Utama Antarmuka Text Editor

 ISSN:2302-8149
e-ISSN:2540-9719

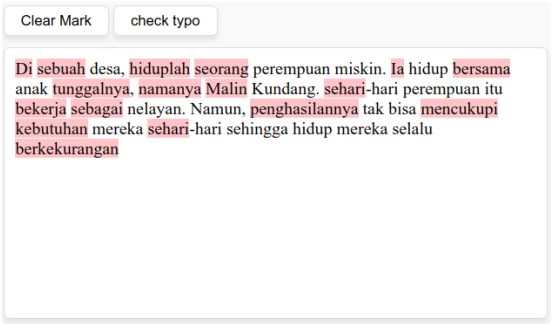
Pada Gambar 6 digambarkan *wireframe* desain utama dari implementasi dari *library*.



Gambar 6. Tampilan Utama Antarmuka *Text* Editor

2. Tampilan Antarmuka Jika Pada Text Editor Ada Kata Yang Salah

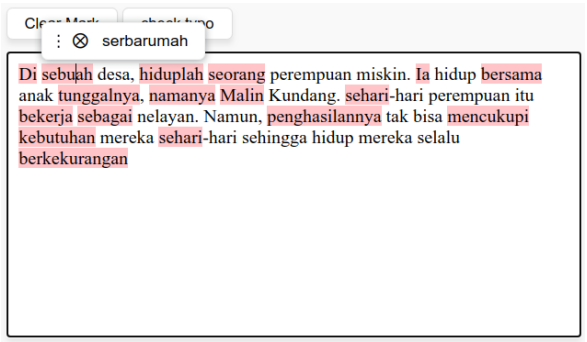
Pada Gambar 7 ditampilkan desain antarmuka jika pada *text* editor ada kata yang salah.



Gambar 7. Tampilan Antarmuka Jika pada *Text* Editor ada kata yang salah

3. Tampilan Rekomendasi Kata Yang Benar

Pada Gambar 8 digambarkan *wireframe* dari tampilan *pop up* yang menampilkan rekomendasi kata yang benar dan juga ada tombol *unmark* dan tombol menu lainnya. Di tombol un mark berguna jika kata yang salah tersebut memang betul katanya tetapi tidak ada pada kamus maka dengan menekan tombol *unmark* akan menghilangkan *mark* merahnya. Sedangkan tombol menu lainnya untuk menampilkan rekomendasi kata yang mirip dengan kata *typo* tersebut.

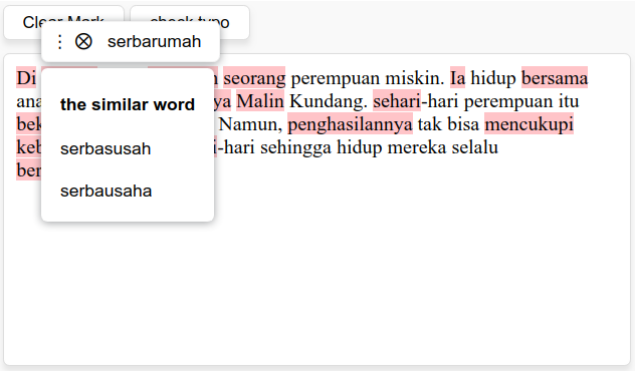


Gambar 8. Tampilan Antarmuka Rekomendasi Kata yang Benar

 ISSN:2302-8149
e-ISSN:2540-9719

4. Tampilan Rekomendasi Kata Lainnya

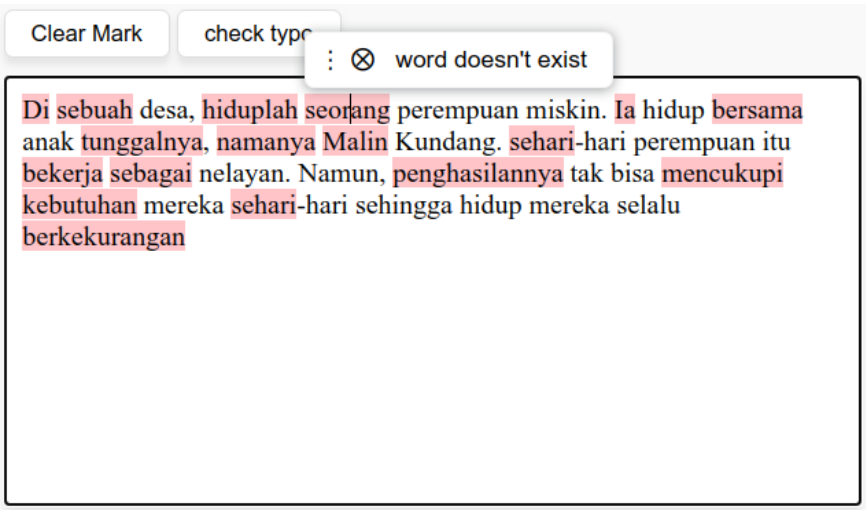
Pada Gambar 9 digambarkan *wireframe* dari tampilan dari tombol menu lainnya. Menampilkan daftar kata yang mirip dengan kata *typo*.



Gambar 9. Tampilan Antarmuka Rekomendasi Kata Lainnya

5. Tampilan Jika Kata Yang Salah Tidak Ada Di Kamus

Pada Gambar 10 ditampilkan *wireframe* dari tampilan jika kata yang salah tidak ada pada kamus maka rekomendasi kata yang dimunculkan di *pop up* yaitu saltikbox akan menampilkan kalimat *word doesn't exist*.



Gambar 10. Tampilan jika Kata yang Salah Tidak ada di Kamus

d. Pengujian Dengan Confusion Matrix dan blackbox

Confusion matrix adalah salah satu metode klasifikasi pengujian untuk mengukur performa dalam suatu uji coba penelitian. Pada tahap pengujian sistem dalam menguji algoritma *Levenshtein Distance* dan metode *regular search expression*, penulis menggunakan *confusion matrix* untuk menguji akurasi dari penerapan metode *regular search expression* dan algoritma *Levenshtein Distance*. Uji coba ini didasari dengan ketersediaan data pada data kamus, penulis juga sudah memandingan satu persatu kata pada potongan cerita secara online di website resmi kamus besar bahasa indonesia. Potongan cerita malin kundang akan diuji coba untuk mengetahui seberapa akurat algoritma dan metode tersebut. Berikut ini adalah potongan cerita malin kundang: Di sebuah desa, hiduplah seorang

 ISSN:2302-8149
e-ISSN:2540-9719

perempuan miskin. Ia hidup bersama anak tunggalnya, namanya Malin Kundang. Sehari-hari perempuan itu bekerja sebagai nelayan. Namun, penghasilannya tak bisa mencukupi kebutuhan mereka sehari-hari sehingga hidup mereka selalu berkekurangan[14].

Berdasarkan uji coba pada algoritma Levenshtein Distance dan metode regular search expression diprediksi terdapat 17 kata yang typo dari 36 kata pada cerita tersebut, dan diantara kata yang typo tersebut hanya 1 kata yang memiliki 3 kata yang mirip dan direkomendasikan untuk menggantikan kata typo tersebut, data hasil *confusion matrix* ada pada Tabel 2.

Tabel 2. Data *Confusion Matrix* Prediksi Kata *Typo*

N = 36	**Aktual**	
Prediksi	**true**	**false**
positif	17	17
negatif	19	0

Setelah mengetahui semua data yang diperlukan, maka akan dihitung *accuracy*, *precision*, dan *recall* dari data tersebut.

a. ***Accuracy***

$$Accuracy = \left(\frac{TP + TN}{TP + FP + FN + TN}\right) * 100\% \quad (2)$$

b. ***Precision***

$$Precision = \left(\frac{TP}{TP + FP}\right) * 100\%. \quad (3)$$

c. ***Recall***

$$Recall = \left(\frac{TP}{(TP + FN)}\right) * 100\%. \quad (4)$$

Tabel 3. Hasil Pengukuran dalam Pencarian Kata *Typo*

Accuracy	**Precision**	**Recall**
67%	50%	100%

Tabel 4. Hasil Pengukuran dalam Kemiripan Kata *Typo*

Accuracy	**Precision**	**Recall**
50%	5%	100%

Berdasarkan rumus persamaan (2-4) didapatkan hasil uji coba dalam Tabel 3 yang telah dilakukan, implementasi algoritma *Levenshtein Distance* dan metode *regular search expression* ini memiliki tingkat akurasi mencapai 67% pada Tabel 4 mencari kata yang *typo* berdasarkan kamus yang disediakan, dan untuk mencari kata yang mirip dengan kata *typo* tersebut memiliki tingkat akurasi mencapai 50%. Adapun beberapa masalah dan penyebab ketidakakuratan yang terjadi di dalam implementasi algoritma dan metode ini adalah sebagai berikut:

a. Metode *regular search expressio*n mencari kata sesuai dengan kata pada kamus yang disediakan. Adapun kata imbuhan seperti “bersama” atau “mencukupi” dan kata gabungan seperti “sehari-hari” tidak ada pada kamus yang sediakan dikarenakan dalam kamus tersebut menyediakan kata tunggal saja.
b. Dan algoritma *Levenshtein Distance* digunakan untuk mencari kata pada kamus yang mirip dengan kata *typo*, adapun 1 kata yang memiliki kemiripan dengan kata pada data kamus yang tersedia yaitu “sebuah” yang memiliki 3 kata yang mirip ”serbarumah”, “serbasusah”, dan “serbausaha”. Adapun 16 kata *typo* yang lainnya tidak mirip, karena dalam pencarian kata pada

 ISSN:2302-8149
e-ISSN:2540-9719

kamus memiliki ketentuan regex untuk mem-*filter* kata yang memiliki panjang sesuai jumlah karakter kata *typo* dan akan ditambahkan 4 agar meningkatkan akurasi dari pencarian kata.

c. Semua sesuai dengan ketentuan data pada kamus yang disediakan, semakin banyak kata yang dimasukkan semakin banyak juga kata yang bisa digunakkan dalam pencarian kata *typo*.

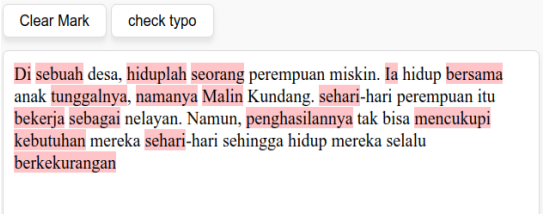
Dalam Pengujian sistem penulis menggunakan metode pengujian *black box* yaitu pengujian yang berfokus pada persyaratan fungsional dari sistem yang digunakan. Adapun metode fungsional dalam pengujian sistem menggunakan metode pengujian *alpha* karena cocok untuk menguji sistem yang baru.

a. Editor

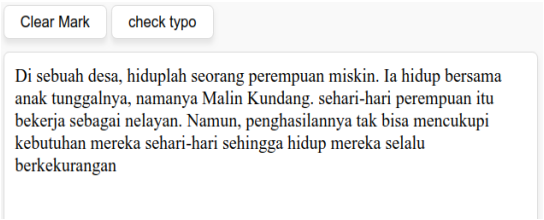
Editor berfungsi untuk menampung teks yang akan dicek kata yang salah dengan rincian data pada Tabel 5 dan gambar 11-12.

Tabel 5. Uji Coba Editor

no	tujuan	input	Output diharapkan	Output sistem
1	Teks editor	Teks : saya cinta amikom	Langsung mendeteksi kata	Tidak terjadi apa apa
2	Klik tombol *check typo*	*Event click*	Langsung mendeteksi kata	Kata yang *typo* akan di *mark* merah
3	Klik tombol *clear*	*Event click*	Hapus teks pada editor	Menghapus mark terhadap kata yang *typo*



Gambar 11. Uji coba editor *marking*



Gambar 12. Uji Coba Editor *Clear Mark*

b. Pop up saltikbox

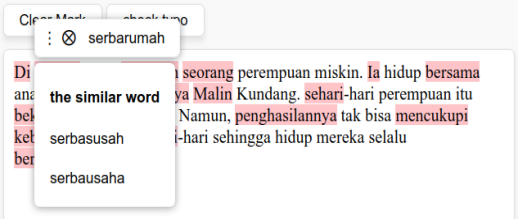
Pop up saltikbox menampung rekomendasi kata yang sesuai dengan kata *typo* dan juga berisi *clear mark* per kata dan rekomendasi kata lainnya yang terdapat pada Tabel 6 dan Gambar 13 di bawah ini.

Tabel 6. Uji Coba *Pop Up* Saltikbox

no	tujuan	input	Output diharapkan	Output sistem
1	Untuk menampilkan	*Event*	Memunculkan kata yang	Memunculkan kata yang

 ISSN:2302-8149
e-ISSN:2540-9719

	rekomendasi kata dan juga *clear mark* per kata	*click*	benar	benar
2	Clear mark per kata	*Event click*	Menghapus mark pada kata	Menghapus mark pada kata
3	Rekomendasi kata lainnya	*Event click*	Memunculkan daftar kata lainnya	Menampilkan daftar kata lainnya



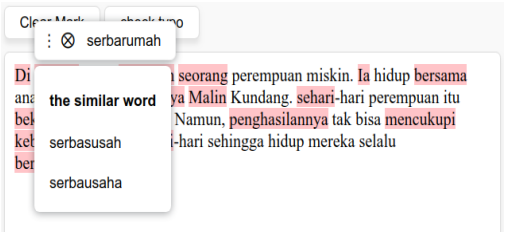
Gambar 13. *Pop Up* Saltikbox

c. Rekomendasi Lata Lainnya

Rekomendasi kata lainnya digunakan untuk menampung kata yang mungkin mirip dengan kata *typo*, seperti ada pilihan kata selain kata yang berada pada *pop up* saltikbox yang terdapat pada Tabel 7 dan Gambar 14 di bawah ini.

Tabel 7. Uji Coba Rekomendasi Kata Lainnya

no	tujuan	input	Output diharapkan	Output sistem
1	Menampilkan daftar kata selain kata pada *pop up* saltikbox	*Event click*	Merubah kata *typo* yang dipilih	Merubah kata *typo* yang dipilih



Gambar 14. Rekomendasi Kata Lainnya

5 Kesimpulan (or Conclusion)

Penggunaan dari algoritma *Levenshtein Distance* metode *regular search expression* telah sukses diimplementasikan kedalam sistem *library* saltikjs. dari beberapa proses dalam sistem, Regex digunakan didalam proses pengolahan kata, proses *searching* kata serupa pada kamus, dan pencarian kata yang *typo* pada *element contenteditable plaintext-only*.Algoritma *Levenshtein Distance* sangat cocok untuk mencari algoritma yang digunakan untuk mencari kesamaan atau kemiripan kata *typo*. Memeriksa 2 kata yang berbeda dan mencari kata yang bisa digunakan sebagai rekomendasi kata untuk mengganti kata yang *typo*.Implementasi metode *regular search expression* dapat digunakan dalam pencarian kata pada teks, dengan menggunakan meta karakter yang ada dan juga fungsi-fungsi yang diperlukan seperti *replace*, *find*, *match*, dan *textContent* dapat membuat metode regex atau *regular search expression* dapat digunakan untuk mencari kata pada teks.Algoritma *Levenshtein*

Distance berhasil diterapkan pada fitur rekomendasi kata pada *library*. Nilai akurasi yang diperoleh mencapai 50% dengan tingkat presisi 5%. Tingkat kemiripan tergantung panjang karakter dari kata *typo* dan ditambahkan 4 untuk menambahkan nilai akurasi yang lebih baik dan juga nilai kemiripan terbesar tergantung dari nilai yang dihasil oleh algoritma *levenshtein distance*. Metode *regular search expression* berhasil diterapkan dan mampu mencari kata pada data kamus yang sudah sediakan. Adapun akurasi yang diperoleh mencapai 67% dan mendapatkan nilai tingkat presisi 50%.Saran penelitian selanjutnya ditambahkan algoritma *stemming* dan *Lemmatization* ke dalam sistem library agar lebih akurat untuk mencocokan kata karena pada *library* hanya memiliki proses sederhana yang menggunakan metode regex dan algoritma *Levenshtein Distance* sebagai dasar dari pencarian kata dan pencarian kemiripan kata dan menambahkan *event* lain seperti *onchange* atau *real time* untuk mengecek secara otomatis, karena pada sistem *library* hanya menggunakan *event* yang *onclick* untuk mengeksekusi proses sistem serta perlu adanya fungsi untuk *library* bisa digunakan ke dalam *plugin* editor editor yang sudah seperti ckeditor, tiny editor, froala, dll.

Referensi

[1] F. Adline, T. Tobing, and R. Nainggolan, "Analisis Perbandingan Penggunaan Metode Binary Search Dengan Regular Search Expression," *METHOMIKA: Jurnal Manajemen Informatika & Komputerisasi Akuntansi*, vol. 4, no. 2, pp. 168–172, 2020, doi: 10.46880/jmika.Vol4No2.pp168-172.

[2] Yogi, I. Ruslianto, and S. Bahri, "Analisa Log Web Server Untuk Mengetahui Pola Perilaku Pengunjung Website Menggunakan Teknik Regular Expressions," *Coding: Jurnal Komputer dan Aplikasi*, vol. 07, no. 01, pp. 120–130, 2019, [Online]. Available: https://httpd.apache.org/docs/2.4/logs.HTM

[3] N. R. Dyah, P. Astuti, F. Noviyanto, and D. Soyusiawati, "Forensik Digital Metode Regex (Regular Expression) Dari Grab Google Search Api Dalam Proses Pelacakan Terhadap Kejahatan Online," *InfoTekjar*, vol. 3, no. 1, pp. 90–94, 2018.

[4] N. Fadhillah, H. Aziz, and D. Lantara, "Validasi Pencarian Kata Kunci Menggunakan Algoritma Levenshtein Distance Berdasarkan Metode Approximate String Matching," *Prosiding Seminar Nasional Ilmu Komputer dan Teknologi Informasi*, vol. 3, no. 2, pp. 129–133, 2018.

[5] Y. Sari, H. Khatimi, and R. Awlia Fajrin, "Deteksi Plagiarisme menggunakan Algoritma Levenshtein Distance," *JTIULM*, vol. 6, no. 1, pp. 31–38, 2021.

[6] R. Adawiyah and N. E. Saragih, "Implementasi Algoritma Levenshtein Distance Dalam Mendeteksi Plagiarisme," *Journal Computer Science and Information Technology(JCoInT) Program Studi Teknologi Informasi*, vol. 2, no. 2, pp. 54–63, 2022.

[7] M. F. Azhri, D. Swanjaya, and R. K. Niswatin, "Penerapan Algoritma Levenshtein Distance pada Aplikasi Asisten Guru Bahasa Inggris," in *Seminar Nasional Inovasi Teknologi*, 2019, pp. 155–160.

[8] Y. Purnama Sari, G. Aditra Pradnyana, and I. Made Agus Wirawa, "Pengembangan Aplikasi Kamus Bahasa Bima-Bahasa Indonesia Menggunakan Algoritma Levenshtein Distance Sebagai Spell Checker Berbasis Android," *Kumpulan Artikel Mahasiswa Pendidikan Teknik Informatika (KARMAPATI)*, vol. 5, no. 2, pp. 86–95, 2001.

[9] E. Apriliansi, A. Lestari, and A. S. Sahay, "Implementasi Algoritma Levenshtein Distance Untuk Pencarian Judul Skripsi Dan Menambahkan Fitur Autocomplete Pada Website Repository Skripsi Jurusan Teknik Informatika Universitas Palangka Raya," *JOINTECOMS (Journal of Information Technology and Computer Science) p-ISSN: 2798-284X*, vol. 1, no. 2, pp. 2798–3862, 2021, doi: 10.47111.

[10] N. Noor Kamala Sari, V. Handrianus Pranatawijaya, P. Bagus Adidyana Anugrah Putra, and P. Studi Teknik Informatika Universitas Palangka Raya Kampus Unpar Tunjung Nyaho Jl Yos Sudarso Palangka Raya, "Penerapan Algoritma Levenshtein distance Untuk Pencarian

ISSN:2302-8149
e-ISSN:2540-9719

Pada Sistem Informasi Perpustakaan Fakultas Kedokteran Universitas Palangka Raya," *JURNAL SAINTEKOM*, vol. 9, no. 1, pp. 66–82, 2019.

[11] A. I. Fahma, I. Cholissodin, and R. Setya Perdana, "Identifikasi Kesalahan Penulisan Kata (Typographical Error) pada Dokumen Berbahasa Indonesia Menggunakan Metode N-gram dan Levenshtein Distance," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 2, no. 1, pp. 53–62, 2017, [Online]. Available: https://www.researchgate.net/publication/323365722

[12] A. Najib, K. B. Utomo, J. T. Informasi, and P. N. Samarinda, "Deteksi Similaritas Dokumen Abstrak Tugas Akhir Menggunakan Metode Levenshtein Distance," *JUST TI*, vol. 10, no. 1, pp. 58–62, 2018.

[13] S. Mariko, "Aplikasi Website Berbasis Html Dan Javascript Untuk Menyelesaikan Fungsi Integral Pada Mata Kuliah Kalkulus," *Jurnal Inovasi Teknologi Pendidikan*, vol. 6, no. 1, pp. 80–91, 2019, doi: 10.21831/jitp.v6.1.22280.

[14] https://dongengceritarakyat.com/, "Dongeng Cerita Rakyat, "Cerita Malin Kundang Singkat – Dongeng Legenda Sumatera Barat," *https://dongengceritarakyat.com/cerita-malin-kundang-singkat-don geng-legenda/*, 2017. Accessed: Jan. 18, 2023. [Online]. Available: https://dongengceritarakyat.com/cerita-malin-kundang-singkat-don geng-legenda/

[15] D. Rosmala and Z. M. Risyad, "Algoritma Levenshtein Distance dalam Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter," *MIND Journal | ISSN*, vol. ISSN, no. 2, pp. 1–12, 2017, doi: 10.26760/mindjournal.

[16] S. Muhammad and S. Azrai, "Pengembangan Sistem Informasi Administrasi Akademik Dengan Rancangan Modul Program Menggunakan Bahasa Pemrograman Berorientasi Objek," in *JOISIE* , 2018, vol. 2, no. 1, pp. 51–57.