

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра Дискретной математики и информационных технологий

РАЗРАБОТКА ТЕКСТОВОГО РЕДАКТОРА С ИСПОЛЬЗОВАНИЕМ
ЭЛЕМЕНТОВ НЕЧЕТКОЙ ЛОГИКИ
КУРСОВАЯ РАБОТА

студента 3 курса 321 группы
направления 09.03.01 — Информатика и вычислительная техника
факультета КНиИТ
Давиденко Алексея Алексеевича

Научный руководитель

Ассистент кафедры ДМиИТ

А.А. Трунов

Заведующий кафедрой

к. ф.-м.н., доцент

Л.Б. Тяпаев

Саратов 2020

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	3
ОПРЕДЕЛЕНИЯ	4
ВВЕДЕНИЕ	5
1 Алгоритмы поиска схожих слов	6
1.1 Расстояние Левенштейна	6
1.2 Расстояние Дамерау-Левенштейна	8
1.3 Metaphone	11
2 Средства для программной реализации	13
2.1 Electron	13
2.2 HTML	14
2.3 Vue	14
3 Реализация алгоритмов	16
3.1 Создание словаря	16
3.1.1 Приведение слова к транслиту	16
3.1.2 Получение русского metaphone	17
3.1.3 Получение словаря	17
3.2 Определение правильного написания слов	18
4 Разработка интерфейса программы	21
4.1 Разработка основного окна приложения	21
4.1.1 Основные моменты разработки	23
4.2 Разработка текстового редактора	24
ЗАКЛЮЧЕНИЕ	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	29
Приложение А Реализация алгоритма транслитерации	31
Приложение Б Реализация русской адаптации metaphone	33
Приложение В Реализация получения словаря	35
Приложение Г Определение правильного написания слов	36
Приложение Д Основные функции интерфейса	42

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

HTML — HyperText Markup Language

CSS — Cascading Style Sheets

JS — JavaScript

ОПРЕДЕЛЕНИЯ

Metaphone — фонетический алгоритм для индексирования слов по их звучанию с учетом основных правил английского произношения;

Фреймворк — универсальная среда разработки программного обеспечения, которая обеспечивает особые функциональные возможности в рамках более большой программной платформы, облегчающая разработку решений;

Редакционное предписание — последовательность действий, необходимая для получения из первой строки второй кратчайшим образом;

ВВЕДЕНИЕ

Нечёткий поиск - это способ поиска информации, которая совпадает шаблону сравнения приблизительно или очень близкого шаблону значения. Алгоритмы нечёткого поиска применяются для распознавания текста, например, при занесении информации с отсканированных документов в базу, нахождения произошедших от некоторого слова слов, в поисковых системах, проверки орфографии и других областях [1].

Проблема нечеткого поиска текстовой информации может заключаться в следующем: имеется некоторый текст. Пользователь вводит в поле поиска запрос, представляющий из себя некоторое слово или последовательность слов, для которых необходимо найти в тексте все совпадения с запросом с учетом всех возможных допустимых различий. Например, при запросе "polynomial" нужно найти также слово "exponential".

Для оценки сходства двух слов в тексте используются специальные метрики нечеткого поиска, которые определяются как минимальное количество односимвольных операций (вставки, удаления, замены), необходимых для превращения одной строки в другую. В качестве метрик используются сходство Джардо-Винклера, расстояние Хемминга, расстояния Левенштейна и Дамерау-Левенштейна и другие.

Целью курсовой работы является реализация алгоритма нахождения расстояния Дамерау-Левенштейна для двух слов, поиск возможности минимизировать время поиска схожих слов и использование полученных алгоритмов в приложении.

Для достижения цели необходимо решить следующие задачи:

1. Реализовать алгоритмы нахождения схожести двух слов;
2. Создать словарь слов, по которому будет производиться поиск схожих слов;
3. Разработать приложение, текстовый редактор, использующий эти методы и словарь.

1 Алгоритмы поиска схожих слов

Для того, чтобы искать слова, в которых допущены ошибки, нужно выбрать метрику, которая будет удовлетворять следующим требованиям:

- Высокая скорость выполнения;
- Малые затраты памяти;
- Точное определение минимального расстояния между словами.

А так же, для того чтобы ускорить и сделать более точным поиск, необходимо производить поиск не только по самим словам, а по их фонетическим кодам. Для этого можно находить слова, расстояние между фонетическими кодами которых менее единицы.

1.1 Расстояние Левенштейна

Рассмотрим следующую задачу: имеется две строки $s1$ и $s2$, необходимо перевести либо $s1$ в $s2$, либо $s2$ в $s1$, используя операции:

- i: Вставка символа в произвольное место;
- d: Удаление символа с произвольной позиции;
- r: замена символа на другой.

Расстоянием Левенштейна для перевода $s1$ в $s2$ для рассматриваемой задачи будет $d(s1, s2)$ - минимальное количество операций i/d/r для перевода $s1$ в $s2$, а редакционное предписание - перечисление операций для перевода с их параметрами.

Для расстояния Левенштейна справедливы следующие утверждения:

1. $d(s1, s2) \geq \|s1\| - \|s2\|$
2. $d(s1, s2) \leq \max(|s1|, |s2|)$
3. $d(s1, s2) = 0 \iff s1 = s2$

, где $|s|$ - это длина строки s .

Искомое расстояние формируется через вспомогательную функцию $D(m, n)$, находящую редакционное расстояние для срезов $s1[0, m]$ и $s2[0, n]$, где $D(i, j)$ находится по формуле 1

$$D(i, j) = \begin{cases} 0 & ; i = 0, j = 0 \\ i & ; j = 0, i > 0 \\ j & ; i = 0, j > 0 \\ D(i-1, j-1) & ; s1[i] = s2[j] \\ \min \begin{pmatrix} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{pmatrix} + 1 & : j > 0, i > 0, s1[i] \neq s2[j] \end{cases} \quad (1)$$

, где (i, j) - клетка матрицы, в которой мы находимся на данном шаге. [2]

Здесь $D(i, 0) = i$ и $D(0, j) = j$ получены из соображения, что любая строка может получиться из пустой, добавлением нужного количества нужных символов, любые другие операции будут только увеличивать оценку.

В общем случае имеем $D(i, j)$ по формуле 2 [2]

$$D(i, j) = D(i-1, j-1), s1[i] = s2[j], \quad (2)$$

иначе по формуле 3

$$D(i, j) = \min \begin{pmatrix} D(i, j-1) \\ D(i-1, j) \\ D(i-1, j-1) \end{pmatrix} + 1 \quad (3)$$

.

В данном случае мы выбираем наиболее выгодную операцию: удаление символа ($D(i-1, j)$), добавление ($D(i, j-1)$) или замена ($D(i-1, j-1)$).

Вычисление матрицы D можно произвести, используя следующий псевдокод:

```

1 input: strings a[1..length(a)], b[1..length(b)]
2 output: distance, integer
3
4 let d[0..length(a), 0..length(b)] be a 2-d array of integers, dimensions
    length(a)+1, length(b)+1
5 // note that d is zero-indexed, while a and b are one-indexed.
6
7 for i := 0 to length(a) inclusive do
```

```

8      d[i, 0] := i
9  for j := 0 to length(b) inclusive do
10     d[0, j] := j
11
12  for i := 1 to length(a) inclusive do
13     for j := 1 to length(b) inclusive do
14        if a[i] = b[j] then
15            cost := 0
16        else
17            cost := 1
18        d[i, j] := minimum(d[i-1, j] + 1,      // deletion
19                           d[i, j-1] + 1,      // insertion
20                           d[i-1, j-1] + cost)  // substitution
21  return d[length(a), length(b)]

```

Результат работы алгоритма для слов “Пять” и “Семь” представлен в таблице 1.1.

Таблица 1.1 – Пример работы алгоритма

		П	Я	Т	Ь
	0	1	2	3	4
С	1	1	2	3	4
Е	2	2	2	3	4
М	3	3	3	3	4
Ь	4	4	4	4	3

1.2 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является модификацией расстояния Левенштейна - к операциям замены, удаления и вставки добавляется операция перестановки символов.

Большинство опечаток могут быть получены из правильного написания по нескольким простым правилам. Дамерау указывает, что 80 процентов всех орфографических ошибок являются результатом [3]:

- Перестановки двух букв;
- Добавления буквы;
- Удаления буквы;
- Написания неправильной буквы.

Для нахождения расстояния Дамерау-Левенштейна используется два алгоритма: упрощенный алгоритм, в котором предполагается, что любой срез

может быть редактирован не более одного раза, и корректный алгоритм, в котором этого ограничения нет.

В обоих алгоритмах редакционное предписание имеет вид:

$$D(i, j) = \min \begin{cases} 0 & ; i = 0, j = 0 \\ D(i - 1, j) + 1 & ; i > 0 \\ D(i, j - 1) + 1 & ; j > 0 \\ D(i - 1, j - 1) + 1_{s1[i] \neq s2[j]} & ; i > 0, j > 0 \\ D(i - 2, j - 2) + 1 & ; i > 1, j > 1, a[i] = b[j - 1], \\ & a[i - 1] = b[j] \end{cases}$$

Различие между алгоритмом нахождения расстояния Левенштейна и упрощенным алгоритмом нахождения расстояния Дамерау-Левенштейна состоит в добавлении условия перестановки в последнем. Псевдокод алгоритма:

```

1 input: strings a[1..length(a)], b[1..length(b)]
2 output: distance, integer
3
4 let d[0..length(a), 0..length(b)] be a 2-d array of integers, dimensions
    length(a)+1, length(b)+1
5 // note that d is zero-indexed, while a and b are one-indexed.
6
7 for i := 0 to length(a) inclusive do
8     d[i, 0] := i
9 for j := 0 to length(b) inclusive do
10     d[0, j] := j
11
12 for i := 1 to length(a) inclusive do
13     for j := 1 to length(b) inclusive do
14         if a[i] = b[j] then
15             cost := 0
16         else
17             cost := 1
18         d[i, j] := minimum(d[i-1, j] + 1,      // deletion
19                             d[i, j-1] + 1,      // insertion
20                             d[i-1, j-1] + cost) // substitution
21         if i > 1 and j > 1 and a[i] = b[j-1] and a[i-1] = b[j] then
22             d[i, j] := minimum(d[i, j],
23                                 d[i-2, j-2] + 1) // transposition
24 return d[length(a), length(b)]

```

Корректный алгоритм находит расстояние без ограничения возможных

перестановок в подстроках, из-за этого ему необходим дополнительный параметр - размер алфавита Σ , такой, что все буквы данных строк будут находиться в массиве $[0, |\Sigma|]$ [4]. Псевдокод алгоритма:

```

1 input: strings a[1..length(a)], b[1..length(b)]
2 output: distance, integer
3
4 da := new array of |Sigma| integers
5 for i := 1 to |Sigma| inclusive do
6     da[i] := 0
7
8 let d[-1..length(a), -1..length(b)] be a 2-d array of integers, dimensions
    length(a)+2, length(b)+2
9 // note that d has indices starting at -1, while a, b and da are one-indexed.
10
11 maxdist := length(a) + length(b)
12 d[-1, -1] := maxdist
13 for i := 0 to length(a) inclusive do
14     d[i, -1] := maxdist
15     d[i, 0] := i
16 for j := 0 to length(b) inclusive do
17     d[-1, j] := maxdist
18     d[0, j] := j
19
20 for i := 1 to length(a) inclusive do
21     db := 0
22     for j := 1 to length(b) inclusive do
23         k := da[b[j]]
24         l := db
25         if a[i] = b[j] then
26             cost := 0
27             db := j
28         else
29             cost := 1
30             d[i, j] := minimum(d[i-1, j-1] + cost, //substitution
31                               d[i, j-1] + 1, //insertion
32                               d[i-1, j] + 1, //deletion
33                               d[k-1, l-1] + (i-k-1) + 1 + (j-l-1)) //
                transposition
34 da[a[i]] := i
35 return d[length(a), length(b)]

```

Здесь da представляет собой массив, хранящий в себе индексы последних совпадений $s1[*]$ с $s2[j]$ ($i' < i$, j : $s1[i'] = s2[j]$), db - индексы последних совпадений $s2[*]$ с $s1[i]$ ($i, j' < j$: $s2[j'] = s1[i]$) .

1.3 Metaphone

Для того, чтобы уменьшить время поиска похожих слов, можно предварительно воспользоваться одним из фонетических алгоритмов.

Фонетические алгоритмы сопоставляют словам с похожим произношением одинаковые коды, что позволяет производить поиск таких слов на основе их фонетического сходства [5].

Например, слова 'Desert' и 'Dessert' будут иметь схожие фонетические коды.

Одним из таких алгоритмов является Metaphone. Этот алгоритм преобразует слова к кодам переменной длины, состоящим только из букв, по сложным правилам. Алгоритм включает в себя следующие шаги [6]:

1. Удаление повторяющихся соседних букв кроме буквы С;
2. Если слово начинается с 'KN', 'GN', 'PN', 'AE', 'WR', убирается первая буква ('KN' -> 'N', 'GN' -> 'N', ...);
3. Опускается 'В' в 'MB', если 'MB' - суффикс;
4. 'С' преобразуется в 'Х', если за ним следует 'IA' или 'H' (только если он не является частью '-SCH-', в этом случае он преобразуется в 'К'). 'С' преобразуется в 'S', если за ним следуют 'I', 'E' или 'Y'. В остальных случаях 'С' преобразуется в 'К';
5. 'D' преобразуется в 'J', если за ним следуют 'GE', 'GY' или 'GI'. В противном случае 'D' преобразуется в 'Т';
6. Удаляется 'G', если за ним следует 'H', причем 'H' стоит не в конце слова и не перед гласным. Также удаляется 'G', если за ним следует 'N' или 'NED', и он является окончанием;
7. 'G' преобразуется в 'J', если до 'I', 'E' или 'Y', и это не в 'GG'. В противном случае 'G' преобразуется в 'К';
8. Опускается 'H', если он стоит после гласного и не перед гласным;
9. 'СК' преобразуется в 'К';
10. 'РН' преобразуется в 'F';
11. 'Q' преобразуется в 'К';
12. 'S' преобразуется в 'Х', если за ним следуют 'H', 'IO' или 'IA';
13. 'Т' преобразуется в 'Х', если за ним следует 'IA' или 'IO'. 'ТН' преобразуется в '0'. 'Т' опускается, если за ним следует 'CH';
14. 'V' преобразуется в 'F';

15. 'WH' преобразуется в 'W', если он стоит в начале. 'W' опускается, если за ним не следует гласная;
16. «X» преобразуется в «S», если он стоит в начале. В противном случае «X» преобразуется в «KS»;
17. 'Y' опускается, если за ним не следует гласная;
18. 'Z' преобразуется в 'S';
19. Опускаются все гласные, кроме начального.

В последствие алгоритм Metaphone был улучшен, была выпущена вторая версия алгоритма, которая получила название Double Metaphone, в которой, в отличие от первой версии, применимой только к английскому языку, учитывалось происхождение слов, особенности их произношения [6]. Для таких слов результатом работы являются два кода - основной вариант произношения и альтернативный [5]. Алгоритм Double Metaphone сложнее, чем его предшественник, увидеть его можно в статье Лоуренса Филиппса 'The double metaphone search algorithm' <https://dl.acm.org/doi/10.5555/349124.349132>.

Пример работы алгоритма: 'My String' будет преобразовано к 'MSTRNK'.

Алгоритм Metaphone был адаптирован к русскому языку. Для русского языка алгоритм состоит из пяти шагов [6]:

1. Преобразование гласных путем следующих подстановок: О, Ы, Я -> А; Ю -> У; Е, Ё, Э, ЁО, ЁЕ -> И;
2. Оглушение согласных букв, за которыми следует любая согласная, кроме Л, М, Н или Р, либо согласных на конце слова путем следующих подстановок: Б -> П; З -> С; Д -> Т; В -> Ф; Г -> К;
3. Удаление повторяющихся букв;
4. Преобразование суффикса слова путем следующих подстановок: УК, ЮК -> 0; ИНА -> 1; ИК, ЕК -> 2; НКО -> 3; ОВ, ЕВ, ИЕВ, ЕЕВ -> 4; ЫХ, ИХ -> 5; АЯ -> 6; ЫЙ, ИЙ -> 7; ИН -> 8; ОВА, ЕВА, ИЕВА, ЕЕВА -> 9; ОВСКИЙ -> @; ЕВСКИЙ -> #; ОВСКАЯ -> \$; ЕВСКАЯ -> %;
5. Удаление букв Ъ, Ь и дефиса.

Из-за небольшого числа правил, адаптированный для русского языка алгоритм Metaphone не отождествляет некоторые схожие фонетически слова.

2 Средства для программной реализации

Для написания программы, которая бы реализовала алгоритмы, описанные выше, нужно, в первую очередь, выбрать подход, которого мы будем придерживаться при разработке. Есть два подхода: разработка нативных приложений, т.е. приложений, которые работают только на определённой платформе или на определённом устройстве, и разработка кроссплатформенных приложений, т.е. приложений, которые способны работать с двумя и более платформами.

В следствие чего был выбран кроссплатформенный подход, т.к. он позволяет разрабатывать программу на одной операционной системе и знать, что разработанная программа будет работать и на других ОС, необходимо лишь перекомпилировать приложение для нужной платформы.

Выбирал я между двумя фреймворками языков, Qt (C++) и Electron (JS), с помощью которых можно писать кроссплатформенные приложения, и, для того чтобы определиться, какой фреймворк использовать, необходимо рассмотреть несколько критериев:

- Лёгкость поддержки приложения;
- Легкость разработки приложения;
- Распространённость фреймворка.

В итоге Electron, оказался наиболее подходящим для разработки и поддержания.

2.1 Electron

Electron - это фреймворк для разработки настольных кроссплатформенных приложений с использованием HTML, CSS и JS [7]. Его особенность состоит в том, что если ты знаешь как разрабатывать сайты, например, то ты сможешь разработать и настольное приложение. По сути, приложение, написанное на Electron представляет собой окно браузера, в котором открыто единственное окно — ваше приложение.

Процесс разработки на Electron разбит на две взаимно зависимые части: разработка интерфейса приложения (фронтэнда) и разработку логической части приложения (бекэнда). Обе эти части можно написать используя лишь JS, HTML и CSS.

2.2 HTML

HTML - это стандартизированный язык гипертекстовой разметки документов во Всемирной паутине. Браузер может интерпретировать описанный с помощью HTML документ и отобразить его структуру на экране пользователя.

После загрузки веб-страницы, браузер создаёт DOM - Document Object Model - объектную модель документа этой страницы. Благодаря этой модели, содержимое сайта можно прочитать и изменить с помощью скриптов, в частности, с помощью JS. Благодаря этому можно описать данные в виде набора утверждений и формул, изменение которых ведет к автоматическому перерасчёту всех зависимостей, сделать сайт реактивным с помощью, например, JS.

Во многих фронтэнд-фреймворках реализована реактивность. Я выбрал VueJS как один из самых популярных и прогрессивных фреймворков.

2.3 Vue

VueJS позволяет декларативно отображать данные в DOM с помощью простых шаблонов. Например, следующий пример кода создаст в DOM компонент, содержащий приветствие:

```
1 <div id = "app">
2   {{message}}
3 </div>
4
5 var app = new Vue ({
6   el: "#app",
7   data: {
8     message: "Hello"
9   }
10 })
```

Данные и DOM теперь реактивно связаны - при изменении данных, DOM автоматически перестроится.

В Vue каждое поле данных автоматически разбивается на пары геттер и сеттер. С их помощью Vue может следить, какие данные читались или изменялись и может определить, какие факторы влияют на отрисовку отображения.

Каждому экземпляру компонента приставлен связанный с ним экземпляр наблюдателя, который помечает все поля, затронутые при отрисовке,

как зависимые. Когда вызывается сеттер поля, помеченного как зависимость, этот сеттер уведомляет наблюдателя, который, в свою очередь, инициирует повторную отрисовку компонента [8].

С помощью этого можно разрабатывать крупные проекты, не отвлекаясь на проблему синхронизации данных.

3 Реализация алгоритмов

3.1 Создание словаря

Перед тем, как реализовать алгоритмы нахождения схожести слов, было необходимо перед этим создать словарь, по которому и будет определяться схожесть слов и предлагаться, если необходимо, варианты замены слов.

Сначала нужно было либо создать, либо скачать готовый сборник русских слов. Я выбрал второй вариант и поэтому скачал с сайта “Архивы форума "Говорим по-русски"” частотный словарь русской литературы.

Затем, с помощью команды `file -i litc-win.txt` определил исходный формат скачанного файла и с помощью `iconv -f iso-8859-1 -t UTF-8 -o litc-win.txt litc-win.txt` изменил кодировку на UTF-8.

Затем выполнил преобразования файла, которые удалили из него слова с частотой менее 1, удалил из всех строк значения частот.

Далее была спроектирована структура словаря, где каждому слову соответствовал объект:

- `word` - само слово;
- `translit` - транслит слова;
- `metaphone` - результат выполнения алгоритма Double Metaphone для транслита слова;
- `cyrMetaphone` - результат выполнения русской адаптации алгоритма Metaphone для слова.

Ввиду сложности реализации алгоритма Double Metaphone, реализация алгоритма была добавлена в виде внешнего пакета. Подробнее эту реализацию можно посмотреть по ссылке [9].

Реализация остальных алгоритмов будет представлена в следующих параграфах.

3.1.1 Приведение слова к транслиту

Для приведения слова к транслиту, необходимо сначала определиться, какой системы транслитерации придерживаться. Я выбрал систему транслитерации международных телеграмм [10] наиболее подходящую для словаря.

Алгоритм приведения слова к транслитерации:

1. Получение входного слова;
2. Создание объекта, содержащего правила транслитерации для каждой

буквы;

3. Инициализация новой строки `newString`, в которую будут добавляться буквы транслитерации слова;
4. Для каждой буквы входного слова:
 - Если в объекте есть ключ, соответствующий очередной букве входной строки, то в `newString` добавляется значение, соответствующее этому ключу;
 - Если в объекте отсутствует ключ, соответствующий очередной букве входной строки в нижнем регистре, то в `newString` добавляется эта буква;
 - Если в объекте есть ключ, соответствующий очередной букве строки в нижнем регистре, то `newChar` присваивается значение, соответствующее этому ключу в верхнем регистре;
5. Полученная строка `newString` подаётся на выход.

Листинг реализации этого алгоритма приведён в приложении А на странице 31, Листинг 2.

3.1.2 Получение русского `metaphone`

Алгоритм русской адаптации `metaphone` можно найти в параграфе 1.3, страница 11.

Листинг реализации этого алгоритма приведен в приложении Б на страница 33, Листинг 3.

3.1.3 Получение словаря

Словарь с предложенной выше структурой можно реализовать с помощью следующего алгоритма:

1. Инициализация пустого словаря `dict`;
2. Чтение данных из текстового файла с массивом слов словаря и частотой использования этих слов;
3. Полученные данные разделить на отдельные строки;
4. В каждой строке удалить частоту использования;
5. Удалить все слова с длиной менее 1;
6. Получить транслитерацию, `double_metaphone` и русский метафон каждого слова;

7. Объединить полученные данные для каждого слова и вывести в новый файл.

Листинг реализации этого алгоритма приведён в приложении В на странице 35, Листинг 5.

Результат работы алгоритма (показаны первые 5 слов из словаря):

Листинг 1: Первые 5 слов полученного словаря

```
1  [  
2    {  
3      "word": "не",  
4      "translit": "ne",  
5      "metaphone": "N",  
6      "cyrMetaphone": "НИ"  
7    },  
8    {  
9      "word": "что",  
10     "translit": "chto",  
11     "metaphone": "КТ",  
12     "cyrMetaphone": "ЧТА"  
13   },  
14   {  
15     "word": "на",  
16     "translit": "na",  
17     "metaphone": "N",  
18     "cyrMetaphone": "НА"  
19   },  
20   {  
21     "word": "он",  
22     "translit": "on",  
23     "metaphone": "АН",  
24     "cyrMetaphone": "АН"  
25   },  
26   {  
27     "word": "как",  
28     "translit": "kak",  
29     "metaphone": "КК",  
30     "cyrMetaphone": "КАК"  
31   },  
32   ...  
33 ]
```

3.2 Определение правильного написания слов

Псевдокод алгоритма Дамерау-Левенштейна приведён в параграфе 1.2, страница 8.

Листинг реализации этого алгоритма приведён в приложении Г на странице 36, Листинг 6.

В предложенном листинге используется вспомогательная функция, которая возвращает не только расстояние Дамерау-Левенштейна, но и отношение расстояния к длине слов и схожесть слов по полученным данным.

Для определения правильности написания слов также была разработана функция, которая получает на вход массив слов (разбитую на слова строку) и возвращает массив объектов, представляющих собой обработанные слова. Структура объектов;

- word - исходное слово;
- correct - правильно ли написано слово;
- suggestions - варианты правильного написания слова, если оно введено неверно;
- spellChecked - вспомогательный параметр, который определяет, проверено слово или нет.

Suggestions из функции возвращаются не в случайном порядке, а в порядке уменьшения схожести с исходным словом.

Также в теле функции находятся транслитерация слова, его double_metaphone и русский metaphone. Это делается для того, чтобы произвести максимально тщательный анализ слов. Алгоритм анализа схожести слов:

1. Находятся translit, double_metaphone и сурMetaphone исходного слова;
2. Каждое слово словаря, которое различается по длине от исходного не более чем на 1, обрабатывается с исходным словом алгоритмом Дамерау-Левенштейна;
3. Если расстояние не больше двух либо схожесть слов хотя бы 0.9, слово словаря добавляется в список возможных вариантов правильного написания исходного слова
4. Полученный массив возможных написаний сортируется по схожести и из функции возвращаются первые 5 возможных написаний.

Листинг реализации этого алгоритма приведён в приложении Г на странице 37, Листинг 7.

Этот алгоритм был протестирован на случайных текстах для определения правильности предложенных исправлений и скорости работы программы. Листинг теста и результаты представлены в в приложении Г на страни-

цах 38, Листинг 8 и 40, Листинг 9 соответственно.

4 Разработка интерфейса программы

Разработку интерфейса я разделил на 2 части:

- Разработка основного окна приложения;
- Разработка остальной части приложения - отдельный клиент текстового редактора.

4.1 Разработка основного окна приложения

Для разработки основного окна использовались VueJS - основной функционал страницы, less - описание стилей страницы.

Для начала с помощью утилиты vue-cli создался каркас окна приложения. Были созданы минимальные компоненты и связи между ними. На рисунке 4.1 представлен изначальный вид vue-приложения.

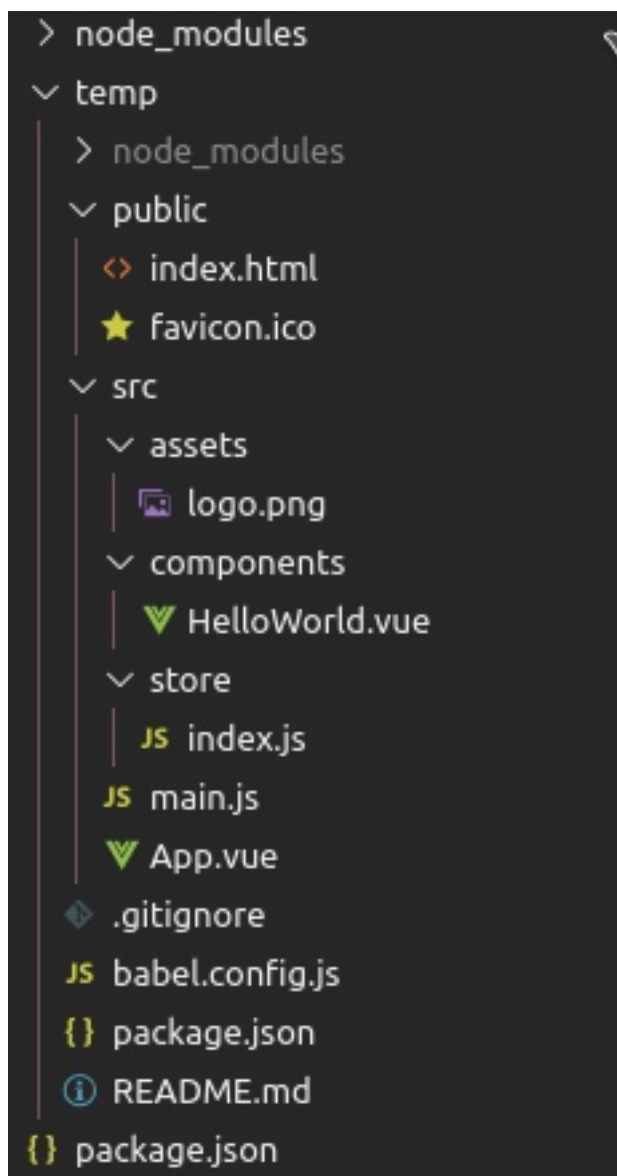


Рисунок 4.1 – Шаблон vue-приложения

Здесь в папке `public` хранятся основной `html`-документ и логотип приложения. В `html`-документе производится подключение основного компонента `app`, указание заголовка и метаданных сайта. На рисунке 4.2 можно увидеть структуру этого `html`-документа.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't
        work properly without JavaScript enabled. Please enable it to continue.</
      strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Рисунок 4.2 – Шаблон `html`-документа

В папке `src` находятся компоненты, папка `components`, файлы `js`-скриптов, `css`, изображений, `assets`, и внутреннего хранилища, `store`.

Внутренне хранилище представляет собой модуль `vuex`, в котором можно хранить общие для проекта данные и получать к ним доступ из любой точки программы. Исходный вид `vuex`-хранилища представлен на рисунке 4.3.

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

export default new Vuex.Store({
  state: {},
  mutations: {},
  actions: {},
  modules: {},
});
```

Рисунок 4.3 – Шаблон vuex-хранилища

В модуле app происходит подключение всех остальных vue-компонентов, находящихся в папке src/components. На рисунке 4.4 представлена структура vue-компонентов.

```
<template>
  <div id="app">
    
    <HelloWorld msg="Welcome to Your Vue.js App" />
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue';

export default {
  name: 'App',
  components: {
    HelloWorld,
  },
};
</script>

<style lang="less">
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

Рисунок 4.4 – Шаблон vue-компонента

4.1.1 Основные моменты разработки

Окно представляет собой массив полей для ввода текста, позади которых располагаются элементы, которые указывают на то, какие слова были

введены неправильно, индикация - красный фон, при наведении указываются варианты замены.

Для текстовых полей были разработаны методы получения списка слов, разделителей слов, методы для объединения строк и создания новых. Также были добавлены функции определения схожести слов с помощью функций помощников.

Были разработаны как синхронные - когда не требуется загрузка больших объемов данных в приложение, так и асинхронные - когда загрузка больших объёмов данных необходима, методы. Пример синхронного метода - поиск по словарю схожих слов; пример асинхронного метода - загрузка словаря в приложение.

Листинги основных файлов можно увидеть в приложении ??.

4.2 Разработка текстового редактора

Для создания клиента был использован electron. Для того, чтобы перевести созданный ранее vue-проект в вид приложения, необходимо добавить vue плагин electron-builder.

После добавления плагина, в папке появился скрипт background.js, который отвечает за отображение разработанного vue-проекта в отдельном приложении. Структуру проекта можно увидеть на рисунке 4.5, а скрипта background.js - на рисунках 4.6 и 4.7.

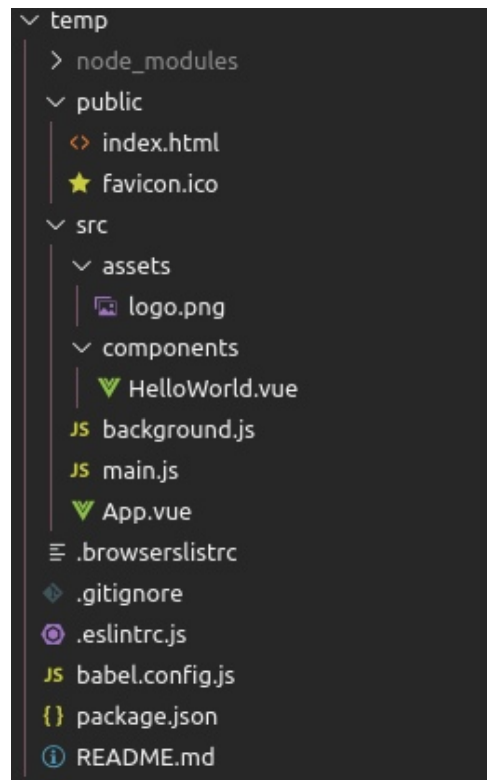


Рисунок 4.5 – Шаблон vue+electron проекта

```
'use strict';

import { app, protocol, BrowserWindow } from 'electron';
import { createProtocol } from 'vue-cli-plugin-electron-builder/lib';
const isDevelopment = process.env.NODE_ENV !== 'production';

let win;

protocol.registerSchemesAsPrivileged([
  { scheme: 'app', privileges: { secure: true, standard: true } },
]);

function createWindow() {
  // Create the browser window.
  win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true,
    },
  });

  if (process.env.WEBPACK_DEV_SERVER_URL) {
    win.loadURL(process.env.WEBPACK_DEV_SERVER_URL);
    if (!process.env.IS_TEST) win.webContents.openDevTools();
  } else {
    createProtocol('app');
    win.loadURL('app://./index.html');
  }

  win.on('closed', () => {
    win = null;
  });
}

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});
```

Рисунок 4.6 – Шаблон background.js, часть 1

```

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', () => {
  if (win === null) {
    createWindow();
  }
});

app.on('ready', async () => {
  if (isDevelopment && !process.env.IS_TEST) {
  }
  createWindow();
});

if (isDevelopment) {
  if (process.platform === 'win32') {
    process.on('message', (data) => {
      if (data === 'graceful-exit') {
        app.quit();
      }
    });
  } else {
    process.on('SIGTERM', () => {
      app.quit();
    });
  }
}
}

```

Рисунок 4.7 – Шаблон background.js, часть 2

Примеры работы приложения представлено на рисунках 4.8 и 4.9.

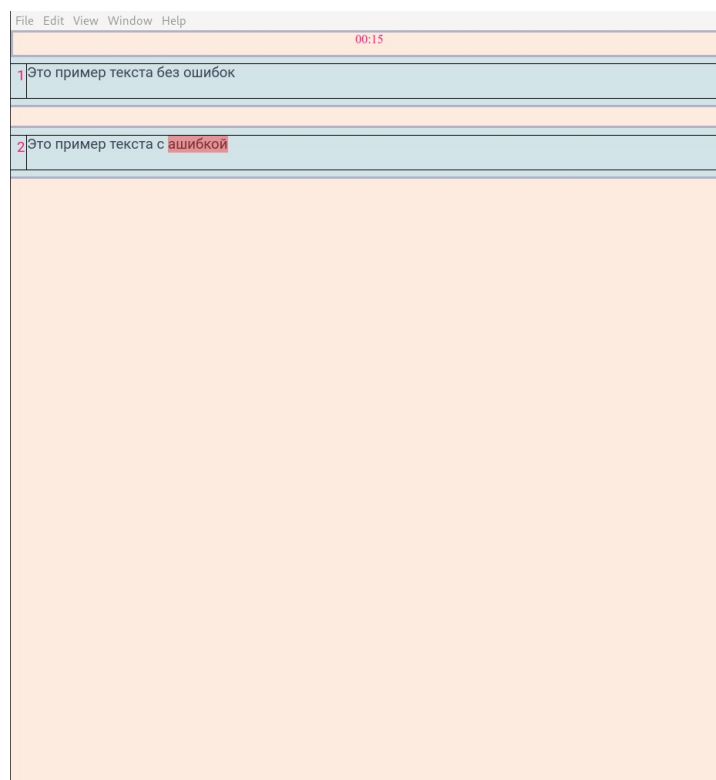


Рисунок 4.8 – Пример 1 работы приложения

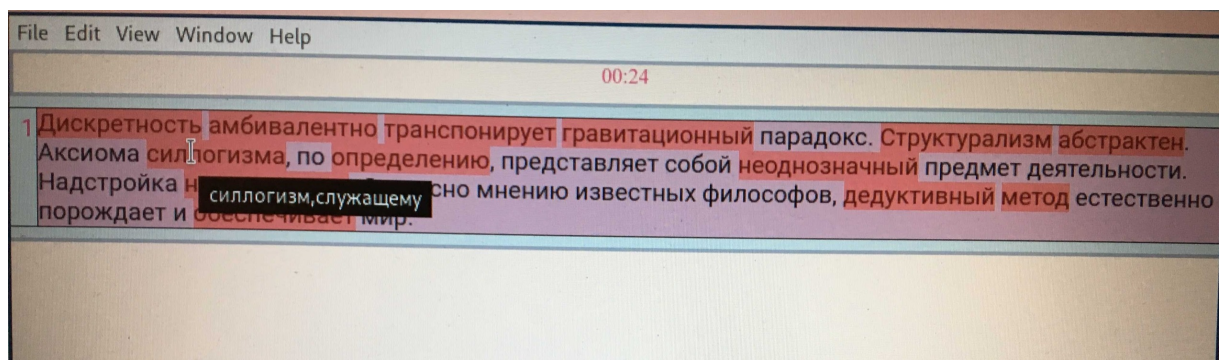


Рисунок 4.9 – Пример 2 работы приложения

ЗАКЛЮЧЕНИЕ

В процессе работы были выявлены качественные превосходства и недостатки использования связки VueJS, ElectronJS и LessJS как платформы для разработки кроссплатформенных реактивных приложений. Был проведён анализ предметной области, включающий в себя изучение алгоритмов схожести слов, поиск актуальных методов решения поставленных задач. Был создан словарь слов, по которому в последствие производился поиск схожих слов и разработано приложение, включающее в себя все разработанные методы.

С полной версией приложения можно ознакомиться по ссылке https://github.com/LoS-Kotyara/COURSE_WORK_3_PROG.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Мосалев П.М. Обзор методов нечеткого поиска текстовой информации [текст] // Вестник МГУП. 2013. №2. URL: <https://cyberleninka.ru/article/n/obzor-metodov-nechetkogo-poiska-tekstovoy-informatsii> (дата обращения: 06.10.2019). Загл. с экрана Яз. рус
- 2 Habr [Электронный ресурс] // <https://habr.com/> – Вычисление редакционного расстояния URL: <https://habr.com/ru/post/117063/> (дата обращения 06.10.2019). Загл. с экрана Яз. рус
- 3 James L. Peterson Computer Programs for Detecting and Correcting Spelling Errors [текст] / James L. Peterson // Communications of the ACM. 1980. №23. С. 676-687. URL: <https://dl.acm.org/doi/10.1145/359038.359041> (дата обращения: 07.10.2019). Загл. с экрана Яз. англ
- 4 Leonid Boytsov Indexing Methods for Approximate Dictionary Searching: Comparative Analysis [текст] / Leonid Boytsov // J. Exp. Algorithmics. 2011. №16. С. 1-93. URL: <https://doi.org/10.1145/1963190.1963191> (дата обращения: 10.11.2019). Загл. с экрана Яз. англ
- 5 Habr [Электронный ресурс] // <https://habr.com/> – Фонетические алгоритмы URL: <https://habr.com/ru/post/114947/> (дата обращения 11.11.2019). Загл. с экрана Яз. рус
- 6 Выхованец Валерий Святославович, Ду Цзяньмин, Сакулин Сергей Александрович Обзор алгоритмов фонетического кодирования [текст] / Выхованец Валерий Святославович, Ду Цзяньмин, Сакулин Сергей Александрович // УБС. 2018. №73. URL: <https://cyberleninka.ru/article/n/obzor-algoritmov-foneticheskogo-kodirovaniya> (дата обращения: 11.11.2019) Загл. с экрана Яз. рус
- 7 Habr [Электронный ресурс] // <https://habr.com/> – Electron: разработка настольных приложений с использованием HTML, CSS и JavaScript URL: <https://habr.com/ru/company/ruvds/blog/436466/> (дата обращения 05.12.2019) Загл. с экрана Яз. рус

- 8 vue.js [Электронный ресурс] // <https://ru.vuejs.org/> – Подробно о реактивности URL: <https://ru.vuejs.org/v2/guide/reactivity.html> (дата обращения 12.12.2019) Загл. с экрана Яз. рус
- 9 npm [Электронный ресурс] // <https://www.npmjs.com/> – double-metaphone URL: <https://www.npmjs.com/package/double-metaphone> (дата обращения 15.12.2019) Загл. с экрана Яз. англ
- 10 Инструкция о порядке обработки международных телеграмм в организациях связи [Текст]: -Москва: [б.и.], 2001. -61 с. (дата обращения 15.12.2019) Загл. с экрана Яз. рус

ПРИЛОЖЕНИЕ А

Реализация алгоритма транслитерации

Листинг 2: Листинг реализации на языке JavaScript

```
1 const toTransliteration = (string: string): string => {
2   interface StringMap {
3     [key: string]: string;
4   }
5
6   let ruEngMatches: StringMap = {
7     а: 'a',
8     б: 'b',
9     в: 'v',
10    г: 'g',
11    д: 'd',
12    е: 'e',
13    ё: 'e',
14    ж: 'j',
15    з: 'z',
16    и: 'i',
17    й: 'i',
18    к: 'k',
19    л: 'l',
20    м: 'm',
21    н: 'n',
22    о: 'o',
23    п: 'p',
24    р: 'r',
25    с: 's',
26    т: 't',
27    у: 'u',
28    ф: 'f',
29    х: 'h',
30    ц: 'c',
31    ч: 'ch',
32    ш: 'sh',
33    щ: 'sc',
34    ъ: '',
35    ы: 'y',
36    ь: '',
37    э: 'e',
38    ю: 'iu',
39    я: 'ia',
40  };
41
42  let newString: string = '';
43  [...string].forEach((char) => {
44    let newChar: string =
```

```
45         ruEngMatches[char] ||
46         (ruEngMatches[char.toLowerCase()] == undefined && char) ||
47         ruEngMatches[char.toLowerCase()].toUpperCase();
48
49         newString += newChar;
50     });
51     return newString;
52 };
```


Реализация русской адаптации metaphone

33

```

25     (str = str
26         .replaceДБВГДЖЗЙКПСТФХЦЧЩ(/(|{|}|}|}|}|}|}|}|}|}/g, 'T$1')
27         .replaceД(/$/ , 'T'));
28
29 export const VtoF = (str: string) =>
30     (str = str
31         .replaceВБВГДЖЗЙКПСТФХЦЧЩ(/(|{|}|}|}|}|}|}|}|}|}/g, 'Φ$1')
32         .replaceB(/$/ , 'Φ'));
33
34 export const GtoK = (str: string) =>
35     (str = str
36         .replaceГБВГДЖЗЙКПСТФХЦЧЩ(/(|{|}|}|}|}|}|}|}|}|}/g, 'K$1')
37         .replaceΓ(/$/ , 'K'));
38
39 export const TStoC = (str: string) => str.replaceТСДС(/|/g, 'Ц');

```

ПРИЛОЖЕНИЕ В

Реализация получения словаря

Листинг 5: Листинг реализации на языке JavaScript

```
1 const makeDictionary = () => {
2   let dict: word[] = [];
3
4   let words = fs
5     .readFileSync(TXT_RU_DICTIONARY_PATH, { encoding: 'utf8' })
6     .split(/\r?\n/)
7     .map((string) => string.replace(/^\s*\d+\s*/, ''))
8     .filter((string) => string.length > 1);
9
10  let transliterated = words.map((word) => funcs.toTransliteration(word));
11
12  let metaphone = transliterated.map((word) => {
13    let temp = doubleMetaphone(word);
14    return temp[0];
15  });
16
17  let cyrMetaphone = words.map((word) => {
18    return funcs.cyrMetaphone(word);
19  });
20
21  for (let i = 0; i < words.length; i++) {
22    dict.push({
23      word: words[iИни],
24      translit: transliterated[i],
25      metaphone: metaphone[i],
26      cyrMetaphone: cyrMetaphone[i],
27    });
28  }
29
30  fs.writeFile(RU_DICTIONARY, JSON.stringify(dict, null, '  '), (err) => {
31    if (err) console.error(err);
32  });
33 };
```

ПРИЛОЖЕНИЕ Г

Определение правильного написания слов

Листинг 6: Листинг реализации алгоритма Дамерау-Левенштейна на языке JavaScript

```
1  const prepare = (steps, length) => {
2    const relative = length == 0 ? 0 : steps / (length - 1);
3    return {
4      steps: steps,
5      relative: relative,
6      similarity: 1 - relative,
7    };
8  };
9
10 const damerauLevenshtein = (str1, str2) => {
11   const str1_length = str1.length;
12   const str2_length = str2.length;
13   const matrix = [];
14   const limit = (str2_length > str1_length ? str2_length : str1_length) + 1;
15   if (Math.abs(str1_length - str2_length) > limit) {
16     return prepare(limit, limit);
17   } else if (str1_length === 0) {
18     return prepare(str2_length, limit);
19   } else if (str2_length === 0) {
20     return prepare(str1_length, limit);
21   }
22   for (let i = 0; i < limit; i++) {
23     matrix[i] = [i];
24     matrix[i].length = limit;
25   }
26   for (let i = 0; i < limit; i++) {
27     matrix[0][i] = i;
28   }
29
30   for (let i = 1; i <= str1_length; ++i) {
31     let str1_i = str1[i - 1];
32     for (let j = 1; j <= str2_length; ++j) {
33       let str2_j = str2[j - 1];
34       let cost = str1_i == str2_j ? 0 : 1;
35       let temp = 0;
36
37       let min = matrix[i - 1][j] + 1; // del
38       if ((temp = matrix[i][j - 1] + 1) < min) min = temp; // ins
39       if ((temp = matrix[i - 1][j - 1] + cost) < min) min = temp; // sub
40       matrix[i][j] =
41         i > 1 &&
42         j > 1 &&
```

```

43         str1_i === str2[j - 2] &&
44         str1[i - 2] === str2_j &&
45         (temp = matrix[i - 2][j - 2] + 1) < min
46         ? temp
47         : min; // trans.
48     }
49 }
50 return prepare(matrix[str1_length][str2_length], limit);
51 };

```

Листинг 7: Листинг реализации алгоритма определения правильности написания слова

```

1 import { cyrMetaphone, toTransliteration } from '../funcs';
2 import { damerauLevenshtein } from '../levenshtein';
3 const doubleMetaphone = require('double-metaphone');
4
5 const sortByKey = (_this, that) => {
6   if (_this.lev.similarity > that.lev.similarity) return 1;
7   if (_this.lev.similarity < that.lev.similarity) return -1;
8   return 0;
9 };
10 let findInObject = (word, dictionary) => {
11   for (let el of dictionary) {
12     if (el.word === word) {
13       return true;
14     }
15   }
16   return false;
17 };
18 const testForSimilarity = (Word, dictionary) => {
19   const word = Word.toLowerCase();
20   const translit = toTransliteration(word);
21   const _word = {
22     word: word,
23     metaphone: doubleMetaphone(translit)[0],
24     cyrMetaphone: cyrMetaphone(word),
25     translit: translit,
26   };
27   let metaphones = [];
28   dictionary.forEach((word) => {
29     if (Math.abs(word.word.length - _word.word.length) > 1) return;
30     const metaphone = damerauLevenshtein(_word.metaphone, word.metaphone);
31     const cyrMetaphone = damerauLevenshtein(
32       _word.cyrMetaphone,
33       word.cyrMetaphone
34     );
35     const wordDif = damerauLevenshtein(word.word.toLowerCase(), _word.word);
36     if (wordDif.steps <= 1 || wordDif.similarity >= 0.9) {

```

```

37     metaphones.push({ word: word.word, lev: wordDif });
38   } else {
39     if (cyrMetaphone.similarity >= 0.9) {
40       metaphones.push({ word: word.word, lev: cyrMetaphone });
41     } else {
42       if (metaphone.similarity >= 0.9) {
43         metaphones.push({ word: word.word, lev: metaphone });
44       }
45     }
46   }
47 });
48 return metaphones
49   .filter((item) => item.word !== word)
50   .sort(sortByKey)
51   .slice(0, 5)
52   .map((word) => word.word);
53 };
54
55
56 const testForSimilarityEntireData = (data, dictionary) => {
57   return data.map((word) => {
58     if (word.length == 1 || findInObject(word.toLowerCase(), dictionary))
59       return {
60         word: word,
61         correct: true,
62         suggestions: [],
63         spellChecked: true,
64       };
65     return {
66       word: word,
67       correct: false,
68       suggestions: testForSimilarity(word, dictionary),
69       spellChecked: true,
70     };
71   });
72 };

```

Листинг 8: Листинг функции тестирования алгоритма

```

1 import { testForSimilarity } from '../testWordForSimilarity';
2 import { normString } from '.';
3
4 import axios from 'axios';
5 import { performance } from 'perf_hooks';
6
7 const testWithRandomRata = async () => {
8   return new Promise(async (resolve) => {
9     let texts: number[] = [];
10    let timings: number[] = [];

```

```

11     for (let i = 0; i < 100; i++) {
12         await axios
13             .get(`https://fish-text.ru/get?type=paragraphnumber=5`)
14             .then((res) => res.data.text)
15             .then((res) => normString(res))
16             .then((data) => {
17                 // console.log(i, data.length);
18                 texts.push(data.length);
19                 let t0 = performance.now();
20                 data.forEach((word) => {
21                     testForSimilarity(word);
22                 });
23                 let t1 = performance.now();
24
25                 // console.log((t1 - t0) / 1000);
26                 timings.push((t1 - t0) / 1000);
27             });
28     }
29     resolve({ texts, timings });
30 });
31 };
32
33 let t1 = performance.now();
34 f().then((res: { texts: number[], timings: number[] }) => {
35     let count = res.texts.length;
36     let text = res.texts.map((el) => {
37         return { 'Words count': el };
38     });
39     let time = res.timings.map((el) => {
40         return {
41             'Time spent': el.toFixed(2) + 's',
42         };
43     });
44     console.table(text);
45     console.table(time);
46
47     console.log(
48         'Time spent processing ' +
49         res.texts.reduce((el, sum) => sum + el, 0) +
50         ' words: ' +
51         ((performance.now() - t1) / 1000).toFixed(2) +
52         's'
53     );
54     let avg = [];
55     for (let i = 0; i < count; i++) {
56         avg.push(res.timings[i] / res.texts[i]);
57     }
58     console.log(

```

```

59     'Average time spent per word: ' +
60     (avg.reduce((a, b) => a + b, 0) / count).toFixed(2)
61   );
62 });

```

Листинг 9: Листинг функции тестирования алгоритма

```

1  +-----+-----+
2  | (index) | Words count |
3  +-----+-----+
4  |    0    |    251     |
5  |    1    |    281     |
6  |    2    |    331     |
7  |    3    |    368     |
8  |    4    |    301     |
9  |    5    |    339     |
10 |    6    |    416     |
11 |    7    |    280     |
12 |    8    |    291     |
13 |    9    |    314     |
14 |   10    |    330     |
15 |   11    |    360     |
16 |   12    |    253     |
17 |   13    |    249     |
18 |   14    |    377     |
19 |   15    |    354     |
20 |   16    |    331     |
21 |   17    |    429     |
22 |   18    |    432     |
23 |   19    |    211     |
24 |   20    |    372     |
25 ...
26 |   99    |    340     |
27 +-----+-----+
28 +-----+-----+
29 | (index) | Time spent  |
30 +-----+-----+
31 |    0    | '9.75s'     |
32 |    1    | '10.12s'    |
33 |    2    | '12.51s'    |
34 |    3    | '13.69s'    |
35 |    4    | '10.75s'    |
36 |    5    | '12.61s'    |
37 |    6    | '14.91s'    |
38 |    7    | '10.42s'    |
39 |    8    | '10.41s'    |
40 |    9    | '11.20s'    |
41 |   10    | '11.81s'    |
42 |   11    | '12.86s'    |

```



```

43 | 12 | '9.40s' |
44 | 13 | '9.48s' |
45 | 14 | '14.68s' |
46 | 15 | '12.46s' |
47 | 16 | '11.27s' |
48 | 17 | '15.50s' |
49 | 18 | '14.36s' |
50 | 19 | '7.46s' |
51 | 20 | '12.10s' |
52 ...
53 | 99 | '13.26s' |
54 +-----+-----+
55 Time spent processing 33489 words: 1237.35s
56 Average time spent per word: 0.04s
57 Done in 1239.44s.

```

ПРИЛОЖЕНИЕ Д

Основные функции интерфейса

Листинг 10: Листинг основного компонента app

```
1 <template>
2   <div id="app">
3     <div id="top-container">
4       <top id="top" />
5     </div>
6     <div id="main-frame-container">
7       <main-frame />
8     </div>
9   </div>
10 </template>
11
12 <script>
13 import mainFrame from './components/mainFrame.vue';
14 import top from './components/top';
15
16 export default {
17   name: 'App',
18   components: {
19     mainFrame,
20     top,
21   },
22   data: function() {
23     return {
24       date: '',
25     };
26   },
27   watch: {
28     date: () => {
29       return new Date().toLocaleString();
30     },
31   },
32 };
33 </script>
34
35 <style lang="less">
36 @import './assets/css/main.less';
37
38 html,
39 body,
40 #app {
41   margin: 0;
42   padding: 0;
43   height: 100%;
44   width: 100%;
```

```

45 }
46
47 #app {
48   display: grid;
49   grid-template-rows: 2rem calc(100vw - 2rem);
50   height: 100vh;
51 }
52 #top {
53   color: @paragraph-lineNumber-color;
54   text-align: center;
55 }
56 #top-container {
57   height: 100%;
58   position: relative;
59
60   border-color: @paragraph-border-color;
61   border-style: solid;
62   border-width: 0.2rem 0.2rem 0rem 0.2rem;
63 }
64 #main-frame-container {
65   height: auto;
66   display: inline;
67 }
68 </style>

```

ЛИСТИНГ 11: ЛИСТИНГ КОМПОНЕНТА mainFrame

```

1 <template>
2   <div class="mainFrame">
3     <div
4       class="paragraph"
5       v-for="(item, index) in textData"
6       :key="index"
7       :ID="index"
8     >
9       <div class="lineNumber">{{ index + 1 }}</div>
10      <div class="textContainer">
11        <div class="backDrop"></div>
12        <textarea
13          class="dataText-textarea"
14          ref="title"
15          type="text"
16          v-on:keyup.alt.down="goTo(index + 1)"
17          v-on:keyup.alt.up="goTo(index - 1)"
18          v-on:input="upgrade(index, $event.target.value, $event)"
19          v-on:keyup.delete="delOrBackPressed(index, $event)"
20          v-bind:value="item"
21          @focus="active(index)"
22          @blur="inactive(index)"

```

```

23         ></textarea>
24     </div>
25 </div>
26 </div>
27 </template>
28
29 <script>
30 export default {
31   name: 'mainFrame',
32   props: {},
33   data() {
34     return {};
35   },
36   computed: {
37     testData() {
38       return this.$store.getters.getValue;
39     },
40
41     strings() {
42       return this.$store.getters.getStrings;
43     },
44   },
45   components: {},
46   updated() {},
47   mounted() {
48     this.$store.dispatch('getDict', { el: this });
49     this.$refs.title[0].focus();
50   },
51   methods: {
52     active(index) {
53       this.$refs.title[index].classList.add('active');
54       this.$refs.title[index].parentElement.children[0].classList.add('
active');
55       this.$refs.title[index].parentElement.classList.add('active');
56     },
57     inactive(index) {
58       this.$refs.title[index].classList.remove('active');
59       this.$refs.title[index].parentElement.children[0].classList.remove(
60         'active'
61       );
62       this.$refs.title[index].parentElement.classList.remove('active');
63     },
64     goTo(index) {
65       this.$nextTick(() => {
66         let input = this.$refs.title[index];
67         input.focus();
68       });
69     },

```

```

70     insertNewParagraph(index) {
71         this.$store.commit('insertNewParagraph', index);
72         this.$nextTick(() => {
73             let input = this.$refs.title[index + 1];
74             input.focus();
75             input.setSelectionRange(0, 0);
76             this.$forceUpdate();
77         });
78     },
79     upgrade: function(index, newValue, e) {
80         // Если введённый символ является Enter
81         if (
82             e.inputType == 'insertLineBreak' ||
83             (e.inputType == 'insertText' && e.data == null)
84         ) {
85             this.insertNewParagraph(index);
86             this.changeLineNumberWidth();
87         }
88
89         // Фильтруем введённые данные
90         // Это необходимо в случаях, когда пользователь скопировал большой
91         // фрагмент текста в поле ввода
92         newValue = newValue.split(/\r?\n/).filter((el) => el);
93
94         // Сохраняем изменённые данные
95         if (newValue.length == 0) {
96             this.$store.commit('editParagraph', {
97                 id: index,
98                 newValue: newValue[0],
99             });
100         } else {
101             for (let i = 0; i < newValue.length; i++) {
102                 this.$store.commit('editParagraph', {
103                     id: index + i,
104                     newValue: newValue[i],
105                 });
106             }
107         }
108
109         let element = this.$refs.title[index];
110
111         // Установка минимального размера поля ввода, если произошло удаление строки
112         element.style.height = 'auto';
113
114         // Установка размера поля ввода равным размеру скроллбара.
115         // Необходимо для автоматического увеличения размера поля
116         element.style.height = element.scrollHeight + 'px';
117

```

```

118     // element.parentElement.children[0].innerHTML = '<div>asd</div>';
119
120     let separators = this.$store.getters.getSeparators(index);
121
122     let div = this.strings[index].map((word, i) => {
123         let suggestions = this.$store.getters.getSuggestions(index, i);
124
125         const spanTemp = `<span title="${suggestions}" onClick="this.
parentNode.parentNode.children[1].focus()" style="display: inline-block;
z-index: 1; "`;
126
127         let element = spanTemp;
128         if (word.correct) element += `class="correct">${word.word}</span>`;
129         else {
130             element += `class="wrong">${word.word}</span>`;
131         }
132         if (separators != undefined && i < separators.length) {
133             element +=
134                 spanTemp + `">${separators[i][0].replace(/\s/g, '&nbsp;')}</span
>`;
135         }
136         return element;
137     });
138
139     element.parentElement.children[0].innerHTML = div.join('');
140 },
141
142 delOrBackPressed(index, e) {
143     let selStart = this.$refs.title[index].selectionStart;
144
145     // Если была нажата кнопка backspace или del при пустой строке,
146     // То значение строки будет неопределённым, а вместе с ним и значение
147     // Длины этой строки. Чтобы обойти эту ситуацию, используется конструкция
148     // try..catch
149     let dataLength;
150     try {
151         dataLength = this.$store.getters.getValueLengthById(index);
152     } catch {
153         dataLength = 0;
154     }
155
156     let input, pos;
157
158     if (e.code == 'Delete') {
159         // Елис была нажата del
160         if (selStart == dataLength) {
161             // Если курсор находится в конце поля ввода
162             input = this.$refs.title[index];

```

```

163         pos = this.$store.getters.getValueLengthById(index);
164
165         const beforeLength = this.$store.getters.getValueLength;
166         this.$store.commit('deleteParagraph', index);
167         const afterLength = this.$store.getters.getValueLength;
168
169         this.changeLineNumberWidth();
170
171         this.$nextTick(() => {
172             if (beforeLength !== afterLength) {
173                 input.focus();
174                 input.setSelectionRange(pos, pos);
175             }
176         });
177     }
178 } else {
179     // Если была нажата backspace
180     const selStart = this.$refs.title[index].selectionStart;
181     if (index > 0 && selStart === 0) {
182         // Если курсор находится в начале поля ввода поля ввода и это поле
183         // Не является пустым
184
185         input = this.$refs.title[index - 1];
186         pos = this.$store.getters.getValueLengthById(index - 1);
187
188         const beforeLength = this.$store.getters.getValueLength;
189         this.$store.commit('backspaceParagraph', index);
190         const afterLength = this.$store.getters.getValueLength;
191
192         this.changeLineNumberWidth();
193
194         this.$nextTick(() => {
195             if (beforeLength !== afterLength) {
196                 input.focus();
197                 input.setSelectionRange(pos, pos);
198             }
199         });
200     }
201 }
202 },
203 changeLineNumberWidth() {
204     this.$nextTick(() => {
205         let dataLength = this.$store.getters.getValueLength;
206         let length = (dataLength + ' ').length;
207
208         this.$el.children.forEach((el) => {
209             el.style.gridTemplateColumns = `${0.25 + length}rem auto`;
210         });

```

```

211     });
212   },
213 },
214 };
215 </script>
216
217 <!-- Add "scoped" attribute to limit CSS to this component only -->
218 <style scoped lang="less">
219 @import '../assets/css/mainframe.less';
220 @import '../assets/css/paragraph.less';
221 </style>

```

Листинг 12: Листинг vuex-хранилища

```

1 import Vue from 'vue';
2 import Vuex from 'vuex';
3
4 /// <reference path="typedefs.js"/>
5
6 import { spellCheck } from '../assets/js/dict';
7
8 import axios from 'axios';
9 let dictionary = '';
10 let getDict = async function() {
11   dictionary = await axios.get('dict.json').then((response) => response.data
12   );
13 };
14 spellCheck;
15 dictionary;
16
17 let flag = false;
18
19 export const UP_TO_DATE = 'up to date';
20 export const OUTDATED = 'outdated';
21 export const EMPTY_WORD = {
22   word: '',
23   correct: true,
24   suggestions: [],
25   status: UP_TO_DATE,
26 };
27
28 /**
29 *
30 * @param {string} string Строка
31 * @returns {string[]} Массив из слов строки
32 */
33 let getWords = (string) => {
34   return string

```



```

35     .replace(/['!""#$%&\\\'()*+,-./:;<=>?@[\\]\^_`{|}~']/g, '')
36     .split(/\s+/)
37     .map((el) => {
38         return {
39             word: el,
40             correct: false,
41             suggestions: [],
42             status: OUTDATED,
43         };
44     });
45 };
46
47 /**
48  *
49  * @param {{
50     words: {
51         word: string;
52         correct: boolean;
53         suggestions: string[];
54         status: string;
55     }[];
56     value: string;
57     preWords: {
58         word: string;
59         correct: boolean;
60         suggestions: string[];
61         status: string;
62     }[];
63 }} string
64 */
65 let makeSpellChecked = (string) => {
66     let spellChecked = spellCheck(
67         string.words.map((word) => word.word),
68         dictionary
69     );
70     let result = {
71         value: string.value,
72         words: spellChecked.map((word) => {
73             return {
74                 word: word.word,
75                 status: UP_TO_DATE,
76                 correct: word.correct,
77                 suggestions: word.suggestions,
78             };
79         }),
80         preWords: string.preWords,
81     };
82     return result;

```

```

83 };
84
85 /**
86  * @param {string} newString - Новая строка
87  * @param {word[]} words - имеющийся массив слов
88  * @returns {word} новый массив слов с изменёнными статусами
89  */
90 let updateData = (newString, words) => {
91   let newWords = newString
92     .replace(/['!"#$%&\\'()*+,-./:;<=>?@[\\]^_`{|}~']/g, '')
93     .split(/\s+/);
94
95   const length = newWords.length;
96
97   for (let i = 0; i < length; i++) {
98     if (!words[i]) {
99       words[i] = {
100         word: newWords[i],
101         correct: false,
102         status: OUTDATED,
103         suggestions: [],
104       };
105
106       continue;
107     }
108     if (words[i].word === newWords[i]) {
109       words[i].status = UP_TO_DATE;
110     } else {
111       words[i].word = newWords[i];
112       words[i].status = OUTDATED;
113     }
114   }
115
116   return words;
117 };
118
119 let getValueLength = (state) => {
120   return state.strings.length;
121 };
122
123 Vue.use(Vuex);
124
125 export default new Vuex.Store({
126   state: {
127     strings: [{ value: '', words: [EMPTY_WORD], preWords: [EMPTY_WORD] }],
128   },
129   getters: {
130     getValueById: (state) => (id) => {

```

```

131         if (state.strings[id]) return state.strings[id].value;
132         return '';
133     },
134     getValueLength: (state) => {
135         return getValueLength(state);
136     },
137     getValue: (state) => state.strings.map((string) => string.value),
138     getValueLengthById: (state) => (id) => state.strings[id].value.length,
139     getPreWords: (state) => (id) => {
140         if (state.strings[id])
141             return state.strings[id].preWords.map((word) => word.word);
142     },
143     getSuggestions: (state) => (id, jd) => {
144         return state.strings[id].words[jd].suggestions;
145     },
146     /**
147      * @param {state} state
148      */
149     getStrings: (state) => {
150         return state.strings.map((string) => string.words);
151     },
152     getSeparators: (state) => (id) => [
153         ...state.strings[id].value.matchAll(/[^a-zA-aZяАЯЁё- -]+/g),
154     ],
155 },
156 mutations: {
157     insertNewParagraph: (state, id) => {
158         state.strings.splice(id + 1, 0, {
159             value: '',
160             words: getWords(''),
161             preWords: getWords(''),
162         });
163     },
164     /**
165      * @param {state} state
166      * @param {{id: number, newValue: string}} object
167      */
168     editParagraph: (state, { id, newValue }) => {
169         if (newValue) {
170             state.strings[id].preWords = state.strings[id].words;
171             state.strings[id].value = newValue;
172
173             state.strings[id].words = updateData(
174                 newValue,
175                 state.strings[id].preWords
176             );
177         } else {
178             state.strings[id].value = '';

```

```

179         state.strings[id].words = getWords(state.strings[id].value);
180         state.strings[id].preWords = state.strings[id].words;
181     }
182 },
183 /**
184  * @param {state} state
185  * @param {id: number} id
186  */
187 deleteParagraph: (state, id) => {
188     if (state.strings[id].value == undefined) {
189         state.strings[id].value = '';
190         state.strings[id].words = getWords(state.strings[id].value);
191         state.strings[id].preWords = state.strings[id].words;
192     }
193
194     try {
195         if (state.strings[id + 1].preWords.map((el) => el.word) == '') {
196             state.strings.splice(id + 1, 1);
197         } else {
198             state.strings[id + 1].words = getWords(state.strings[id + 1].value
199 );
200             let words = state.strings[id + 1].words.map((el) => el.word).join
201 ('');
202             let preWords = state.strings[id + 1].preWords
203                 .map((el) => el.word)
204                 .join('');
205
206             if (
207                 (words == '' && preWords == '') ||
208                 (words == preWords && flag == true) ||
209                 (words == '' && preWords.length >= 1 && flag == true)
210             ) {
211                 state.strings[id].value += state.strings[id + 1].value;
212                 state.strings[id].words = updateData(
213                     state.strings[id].value,
214                     state.strings[id].preWords
215                 );
216                 state.strings[id + 1].value = '';
217                 state.strings[id + 1].words = getWords(state.strings[id + 1].
218 value);
219
220                 state.strings[id + 1].preWords = state.strings[id].words;
221                 flag = false;
222                 state.strings.splice(id + 1, 1);
223             } else if (
224                 (words == preWords && flag != true) ||
225                 (words == '' && flag == false)
226             ) {

```

```

224         flag = true;
225     }
226 }
227 } catch {
228     return;
229 }
230 },
231 /**
232  * @param {state} state
233  * @param {id: number} id
234  */
235 backspaceParagraph: (state, id) => {
236     if (state.strings[id].value == undefined) {
237         state.strings[id].value = '';
238         state.strings[id].words = [EMPTY_WORD];
239         state.strings[id].preWords = [EMPTY_WORD];
240     }
241
242     try {
243         if (state.strings[id].preWords.map((el) => el.word) == '') {
244             state.strings.splice(id, 1);
245         } else {
246             state.strings[id].words = getWords(state.strings[id].value);
247             let words = state.strings[id].words.map((el) => el.word).join('');
248             let preWords = state.strings[id].preWords
249                 .map((el) => el.word)
250                 .join('');
251
252             if (
253                 (words == '' && preWords == '') ||
254                 (words == preWords && flag == true) ||
255                 (words == '' && preWords.length >= 1 && flag == true)
256             ) {
257                 state.strings[id - 1].value += state.strings[id].value;
258                 state.strings[id - 1].words = updateData(
259                     state.strings[id - 1].value,
260                     state.strings[id - 1].preWords
261                 );
262                 state.strings[id].value = '';
263                 state.strings[id].words = getWords(state.strings[id].value);
264
265                 state.strings[id].preWords = state.strings[id].words;
266                 flag = false;
267                 state.strings.splice(id, 1);
268             } else if (
269                 (words == preWords && flag != true) ||
270                 (words == '' && flag == false)
271             ) {

```

```

272         flag = true;
273     }
274 }
275 } catch {
276     return;
277 }
278 },
279 updateStrings: (state, { result }) => {
280     state.strings = result;
281 },
282 },
283 actions: {
284     getDict: (state, { el }) => {
285         getDict();
286         setInterval(() => {
287             let result = state.state.strings.map((string) =>
288                 makeSpellChecked(string)
289             );
290             state.commit('updateStrings', { result });
291             el.$forceUpdate();
292         }, 4000);
293     },
294 },
295 modules: {},
296 });

```