МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Забайкальский государственный университет»

(ФГБОУ ВО «ЗабГУ»)

Факультет энергетический

Кафедра информатики, вычислительной техники и прикладной математики

**КУРСОВАЯ РАБОТА**

По дисциплине Теория автоматов

на тему: «Программная реализация конечных автоматов»

Выполнили студенты гр. ВМК-21

Кувыкин Н.Д

Сизиков А.С

Проверил доцент кафедры ИВТ и ПМ

Коган Е.С.

Чита

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Забайкальский государственный университет»

(ФГБОУ ВО «ЗабГУ»)

Факультет энергетический

Кафедра информатики, вычислительной техники и прикладной математики

ЗАДАНИЕ

на курсовую работу

По дисциплине: Теория автоматов

Студенту: Кувыкину Никите Денисовичу

Студенту: Сизикову Алексею Сергеевичу

Специальности: 09.03.01 Информатика и вычислительная техника

1 Тема курсовой работы: «Программная реализация конечных автоматов»

2 Срок подачи студентом законченной работы: 23 декабря 2024 г.

3 Исходные данные к работе: литературные источники, источники в сети интернет

4 Перечень подлежащих в курсовой работе вопросов:

а) Постановка и анализ задачи.

б) Анализ данных.

в) Программная реализация.

г) Техническое задание.

д) Руководство пользователя.

5 Перечень графического материала (если имеется): -

Дата выдачи задания «3» сентября 2024г.

Руководитель курсовой работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Коган. Е.С. /

(подпись, расшифровка подписи)

Задание принял к исполнению

«9» сентября 2024г.

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Сизиков А.С. /

Подпись студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кувыкин Н.Д. /

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Забайкальский государственный университет»

(ФГБОУ ВО «ЗабГУ»)

Факультет энергетический

Кафедра информатики, вычислительной техники и прикладной математики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по 09.03.01 Информатика и вычислительная техника

на тему: «Программная реализация конечных автоматов»

Выполнил студент группы ВМК-21 Кувыкин Н.Д

Выполнил студент группы ВМК-21 Сизиков А.С

Руководитель работы: доцент кафедры ИВТ и ПМ Коган Евгения Семёновна

РЕФЕРАТ

Пояснительная записка – 38 стр., иллюстраций – 6, приложений – 2.

АВТОМАТ, РАССТОЯНИЕ ЛЕВЕНШТЕЙНА, WEB-СЕРВЕР, РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ, СОСТОЯНИЕ, СОБЫТИЕ.

В работе описывается создание web-сервера для автоисправления с использованием конечного автомата и алгоритма расстояния Левенштейна. Реализация серверной части производится с использованием Flask, а клиентская – с использованием JavaScript. В курсовой работе рассматриваются принципы работы конечного автомата и его применение для реализации функций автоисправления. Также анализируется алгоритм расстояния Левенштейна и его роль в улучшении точности исправлений. Предусмотрено руководство пользователя для взаимодействия с сервером и его возможностями.

[ВВЕДЕНИЕ 6](#_Toc185795391)

[1 Постановка и анализ задачи 7](#_Toc185795392)

[2 Анализ данных 10](#_Toc185795393)

[3 Программная реализация 12](#_Toc185795394)

[4 Техническое задание 18](#_Toc185795395)

[5 Руководство пользователя 20](#_Toc185795401)

[ЗАКЛЮЧЕНИЕ 22](#_Toc185795402)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc185795403)

[Приложение А 24](#_Toc185795404)

[Приложение Б 28](#_Toc185795405)

# ВВЕДЕНИЕ

Термин «автоматы» происходит от греческого слова «αὐτόματα», что означает «самодействующий». Автомат – это абстрактное самодостаточное вычислительное устройство, которое автоматически выполняет заданную последовательность операций. Исследования в данной области начались в середине XX века. Несмотря на свою простоту, модель конечного автомата показывает свою эффективность во многих областях инженерной деятельности. Это объясняется необходимостью создания и эксплуатации современной вычислительной техники, средств передачи и обработки информации, автоматизированных систем управления и проектирования.

Для выполнения курсовой работы произведен анализ основных понятий теории автоматов, чтобы реализовать конечный автомат в программной среде. Необходимо изучить, какие алгоритмы задействуются для выполнения операций, принцип работы конечного автомата.

## 1 Постановка и анализ задачи

**1.1 Описание предметной области**

Конечные автоматы (Finite State Machines, FSM) представляют собой модель вычислений, которая может находиться в одном из конечного числа состояний в каждый момент времени. Эти состояния изменяются в ответ на события, которые могут произойти в системе[3].

Конечные автоматы широко используются в программировании для моделирования поведения систем. Они позволяют абстрагировать сложные процессы и бизнес-логику, предоставляя ясную структуру для управления состоянием приложения.

В контексте регулярных выражений конечные автоматы могут быть использованы для поиска неправильно написанных слов или некорректного постановления орфографических знаков.

Применение конечных автоматов в разработке регулярных выражений, особенно с использованием алгоритма расстояния Левенштейна[1], позволяет разработчикам эффективно управлять сложным состоянием приложения, делая код более структурированным и поддерживаемым. Это также способствует более легкому тестированию и отладке.

Конечные автоматы тесно связаны с теорией регулярных выражений. Каждое регулярное выражение может быть преобразовано в конечный автомат, который затем используется для выполнения операций, таких как поиск или проверка соответствия строк.

Конечные автоматы применяются для следующих задач:

*Поиск подстрок*: применение конечных автоматов позволяет находить все вхождения заданной подстроки в текст, используя эффективные алгоритмы поиска, такие как алгоритм Кнута-Морриса-Пратта.

*Проверка синтаксиса*: регулярные выражения могут определять шаблоны для проверки формата данных, например, телефонных номеров, адресов электронной почты или IP-адресов.

*Обработка ошибок*: с помощью конечных автоматов и расстояния Левенштейна возможно автоматическое исправление орфографических ошибок, например, предложить правильное слово на основе неправильного ввода.

*Разбор текста*: использование конечных автоматов позволяет анализировать текст, разбивая его на токены, что полезно в компиляторах или обработке естественного языка.

**1.2 Постановка задачи**

Целью курсовой работы является разработка приложения, которое будет выполнять функцию автоисправления текста с использованием конечных автоматов и алгоритма расстояния Левенштейна. Это приложение позволит пользователю вводить текст и исправлять орфографические и пунктуационные ошибки, обеспечивая высокую точность и удобство использования.

**1.3 Средства реализации**

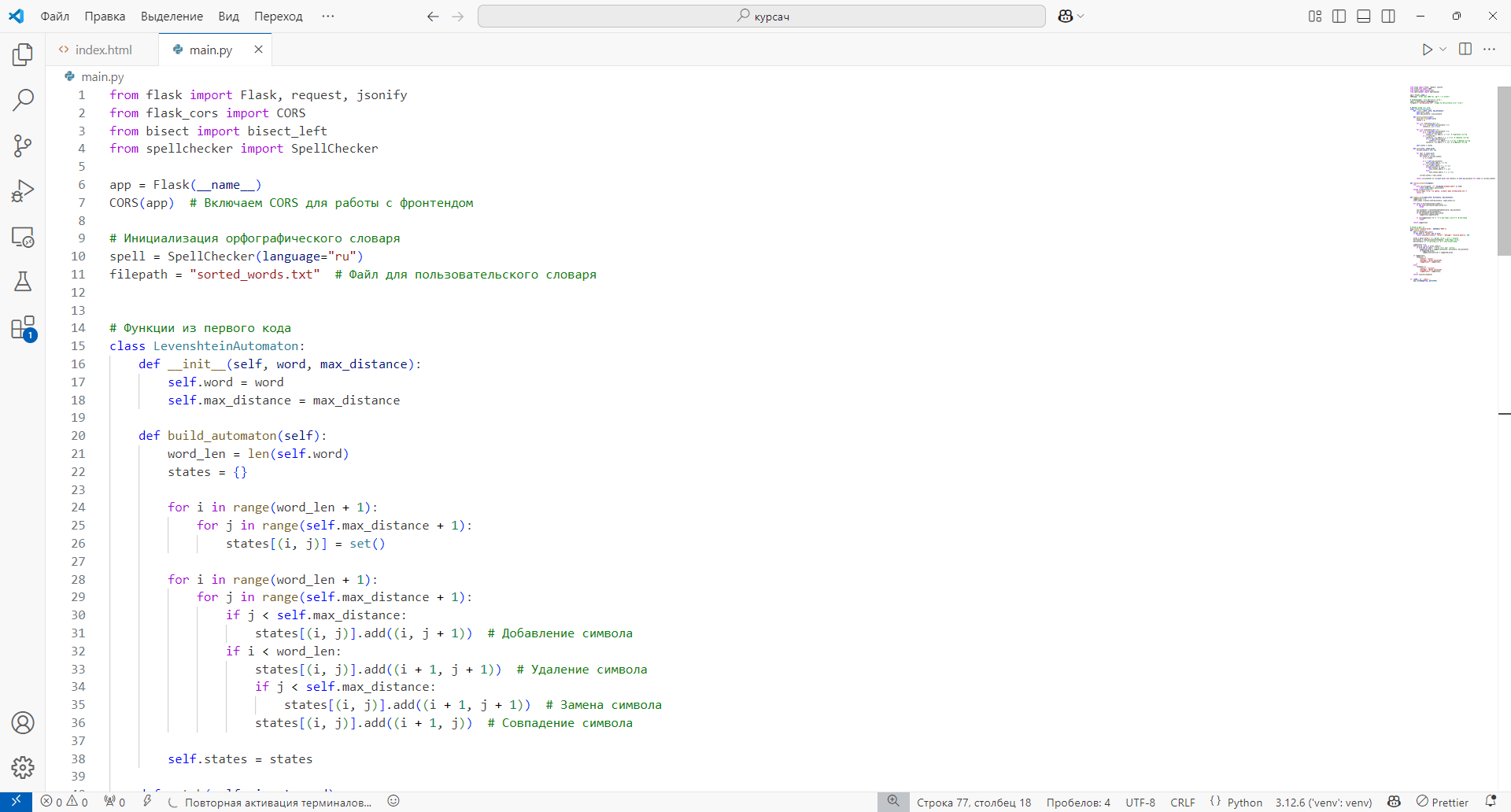
Для реализации выбрана интегрированная среда разработки программного обеспечения Visual Studio Code с использование языка программирования высокого уровня Python, JavaScript. Внешний вид интерфейса указан на рисунке ****

Рисунок 1 – Интерфейс Visual Studio Code

## 2 Анализ данных

Данные приложения делятся на входные, выходные и промежуточные.

**2.1 Входные данные**

Входными данными являются текстовые данные, которые вводит пользователь. Например, это могут быть строки, вводимые через клавиатуру, или текст, вставленный из буфера обмена. Помимо текста, входными данными могут быть события, связанные с действиями пользователя, такие как:

* Нажатие клавиш.
* Выбор опции из выпадающего списка (например, при автозамене слова).

**2.2 Выходные данные**

Выходные данные – это результат обработки текста приложением. К ним относятся:

* Исправленный текст с учетом орфографических и пунктуационных ошибок.
* Варианты замены слова, если оно написано с ошибкой, с указанием степени вероятности.
* Подсказки пользователю, включая рекомендации по расстановке пробелов или знаков препинания.

**2.3 Промежуточные данные**

Промежуточные данные представляют собой информацию, которая используется приложением для обработки текста и управления состояниями конечных автоматов. Эти данные невидимы для пользователя и включают:

* Текущее состояние конечного автомата.
* Промежуточные результаты вычисления расстояния Левенштейна (например, матрицы или частичные подсчеты).
* Буфер анализа текста: выделенные слова, их позиции, текущие операции (вставка, удаление, замена).
* Контекст анализа (например, правила для проверки пунктуации или набор регулярных выражений).

## 

## 3 Программная реализация

**3.1 Интерфейс программы.**

В ходе разработки создан интерфейс веб-приложения. Приложение имеет одно окно с полем ввода текста:

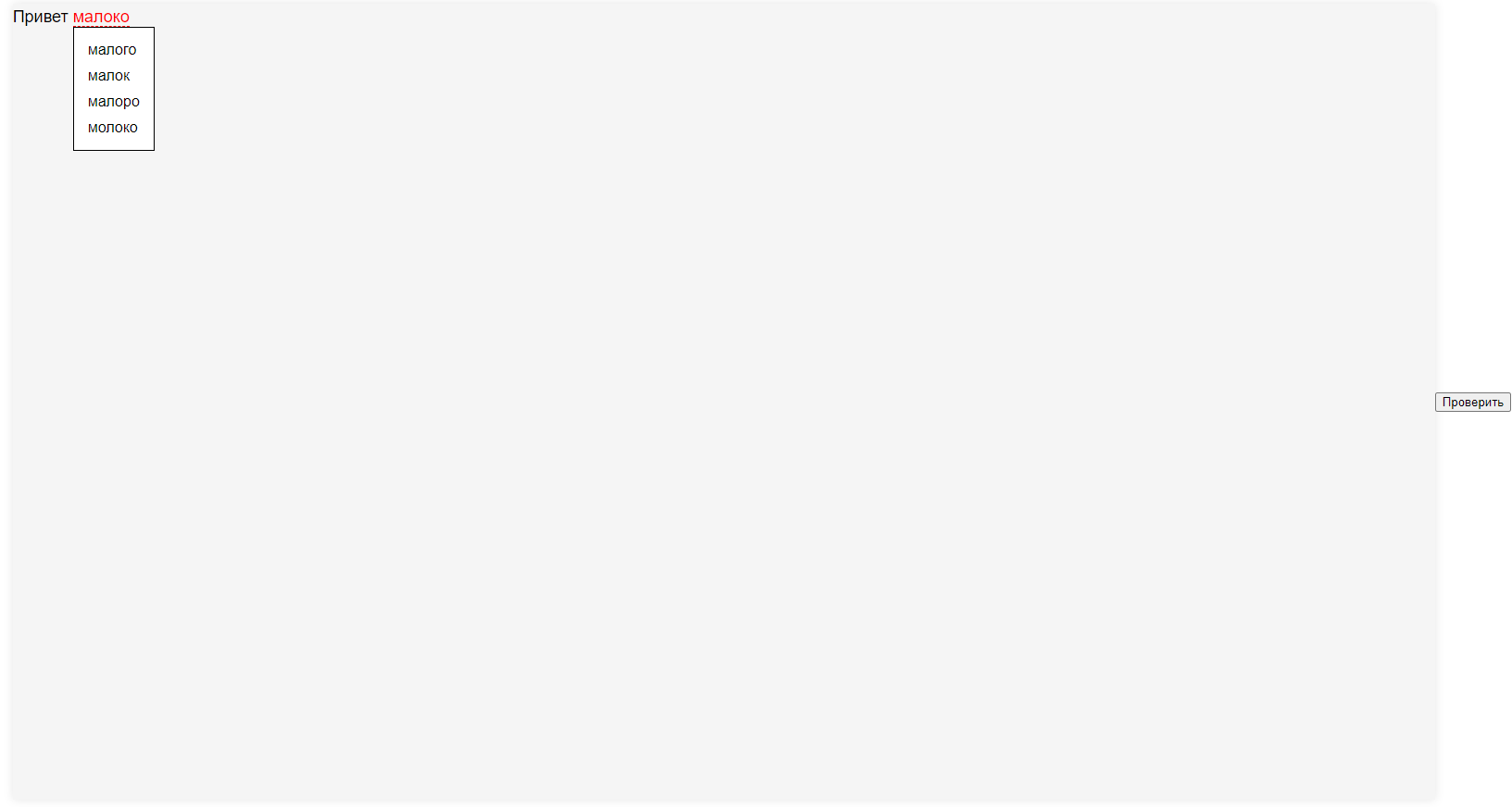


Рисунок 2 – Интерфейс веб-приложения

Код конечного автомата, отвечающего за подсветку пунктуационных ошибок:

    function wrapWordsInSpans(text) {

      const wordsAndPunctuation = text.split(/(\s\*[.,!?]\s\*|\s+)/).filter(Boolean);

      let wrappedText = "";

      let wordCounter = 0;

      let spanCounter = 0;

      wordsAndPunctuation.forEach((part) => {

        if (/^\s\*[.,!?]\s\*$/.test(part) || /^\s+$/.test(part)) {

          const spaceId = `space\_${spanCounter++}`;

          wrappedText += `<span id="${spaceId}">${part}</span>`;

        } else if (part.trim().length > 0) {

          const wordId = `word\_${wordCounter++}`;

          wrappedText += `<span id="${wordId}">${part}</span>`;

        }

      });

      editor.innerHTML = wrappedText;

    }

    function collectSpans() {

      const spanElements = document.querySelectorAll('span');

      const spans = Array.from(spanElements).map(span => ({

        id: span.id,

        text: span.textContent,

        type: span.getAttribute('data-error') ? 'error' : 'normal'

      }));

      return spans;

    }

function checkTextWithFiniteAutomaton(spans) {

  const states = {

    START: 0,

    WORD: 1,

    PUNCTUATION: 2,

    SPACE: 3

  };

  let currentState = states.START;

  let isFirstWord = true;

  let consecutiveSpaces = 0;

  let lastSpaceId = null;

  spans.forEach((span, index) => {

    const nextSpan = spans[index + 1];

    const text = span.text;

    switch (currentState) {

      case states.START:

      case states.PUNCTUATION:

        if (span.id.startsWith('word\_')) {

          if (isFirstWord && !/^[A-ZА-Я]/.test(text)) {

            document.getElementById(span.id).classList.add('error');

            document.getElementById(span.id).setAttribute('data-error', 'Предложение должно начинаться с заглавной буквы');

          }

          isFirstWord = false;

          currentState = states.WORD;

          consecutiveSpaces = 0;

        }

        console.log(currentState);

        break;

      case states.WORD:

        if (span.id.startsWith('space\_')) {

          if (/\s+[.,!?]/.test(text)) {

            document.getElementById(span.id).classList.add('highlight-green');

            document.getElementById(span.id).setAttribute('data-error', 'Лишний пробел перед знаком препинания');

          }

          if (text.trim() === '') {

            consecutiveSpaces++;

            lastSpaceId = span.id;

            if (consecutiveSpaces > 1) {

              document.getElementById(span.id).classList.add('highlight-green');

              document.getElementById(span.id).setAttribute('data-error', 'Несколько пробелов подряд');

            }

          }

          if (/[.!?]/.test(text)) {

            isFirstWord = true;

            currentState = states.PUNCTUATION;

          } else {

            currentState = states.SPACE;

          }

        }

        console.log(currentState);

        break;

      case states.SPACE:

        if (span.id.startsWith('space\_')) {

          const prevSpan = span.id;

          console.log(span.id);

          if (/[.,!?]/.test(prevSpan)) {

            document.getElementById(span.id).classList.add('highlight-green');

            document.getElementById(span.id).setAttribute('data-error', 'Отсутствует пробел после знака препинания');

          }

          currentState = states.WORD;

          consecutiveSpaces = 0;

        }

        console.log(currentState);

        break;

    }

  });

}

Блок кода конечного автомата, отвечающего за расстояние Левенштейна:

class LevenshteinAutomaton:

    def \_\_init\_\_(self, word, max\_distance):

        self.word = word

        self.max\_distance = max\_distance

    def build\_automaton(self):

        word\_len = len(self.word)

        states = {}

        for i in range(word\_len + 1):

            for j in range(self.max\_distance + 1):

                states[(i, j)] = set()

        for i in range(word\_len + 1):

            for j in range(self.max\_distance + 1):

                if j < self.max\_distance:

                    states[(i, j)].add((i, j + 1))  # Добавление символа

                if i < word\_len:

                    states[(i, j)].add((i + 1, j + 1))  # Удаление символа

                    if j < self.max\_distance:

                        states[(i, j)].add((i + 1, j + 1))  # Замена символа

                    states[(i, j)].add((i + 1, j))  # Совпадение символа

        self.states = states

    def match(self, input\_word):

        current\_states = {(0, 0)}

        for char in input\_word:

            next\_states = set()

            for state in current\_states:

                i, j = state

                if j < self.max\_distance:

                    next\_states.add((i, j + 1))

                if i < len(self.word):

                    next\_states.add((i + 1, j + 1))

                    if self.word[i] == char:

                        next\_states.add((i + 1, j))

                    else:

                        next\_states.add((i + 1, j + 1))

            current\_states = next\_states

        return any(state[0] == len(self.word) and state[1] <= self.max\_distance for state in current\_states)

**3.2 Диаграмма состояний.**

Диаграмма конечного автомата, демонстрирующая его работу представлена на рисунке 3.

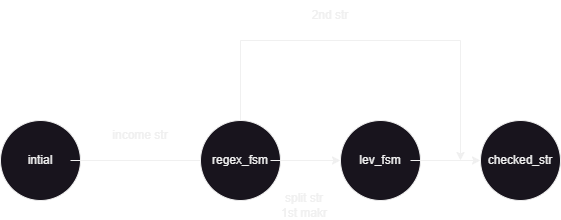


Рисунок 3 – Диаграмма конечного автомат

На диаграмме показан процесс работы приложения для автоисправления текста, где каждый компонент выполняет определенную функцию. В начальном состоянии, обозначенном как initial, приложение ожидает ввод строки от пользователя. Эта строка может быть набрана через клавиатуру или вставлена из буфера обмена. После ввода текст передается в состояние regex\_fsm, где конечный автомат, основанный на регулярных выражениях, обрабатывает строку. На этом этапе выполняется проверка структуры текста, включая корректность пунктуации и пробелов. Также строка разбивается на отдельные слова.

Обработанные части текста передаются в состояние lev\_fsm, где работает конечный автомат расстояния Левенштейна. Он сравнивает каждое слово с доступным словарем, вычисляет расстояние Левенштейна и предлагает исправления для слов, содержащих ошибки. Если слово корректно, оно передается дальше без изменений. Исправленные и проверенные слова затем формируют итоговый результат.

В конечном состоянии checked\_str приложение объединяет все обработанные данные в исправленный текст. Если в процессе проверки были найдены ошибки, они исправляются, и пользователю возвращается скорректированный текст. В случае, если ввод изначально был корректным, текст передается без изменений. Взаимодействие между состояниями может быть циклическим, позволяя повторно обрабатывать текст в случае его изменения пользователем.

## 4 Техническое задание

## 4.1 Введение

Приложение представляет собой веб-сервер, позволяющий пользователю вводить текст в специальное поле ввода. Введенная строка проверяется на наличие орфографических и пунктуационных ошибок, после чего пользователю предлагаются варианты исправлений.

## 4.2 Требования к функциональным характеристикам

Приложение должно предоставлять следующие функции:

* Отображение списка вариантов замены слов с ошибками.
* Подсветка ошибок в тексте для облегчения их идентификации.
* Возможность выбора предложенного исправления или сохранения исходного слова.

## 4.3 Требования к надежности

* Приложение должно корректно работать при вводе больших объемов текста.
* В случае некорректного ввода данных (например, пустой строки или символов, не входящих в алфавит), пользователь должен получать уведомление об ошибке.
* При сбоях или внутренних ошибках приложения необходимо отображать информативное сообщение о проблеме и предлагать варианты ее устранения.

## 4.4 Требования к программной документации

Программная документация должна включать:

* Руководство пользователя, описывающее процесс работы с приложением, основные функции и примеры взаимодействия.

## 4.5 Требования к информационной и программной совместимости

* Приложение должно функционировать на веб-сервере, поддерживающем Python и фреймворк Flask.
* Для корректной работы приложения пользователь должен использовать современный веб-браузер с поддержкой HTML5, CSS3 и JavaScript.

## 5 Руководство пользователя

После загрузки веб-страницы пользователя встретит экран с полем ввода текста, как на рисунке 4.

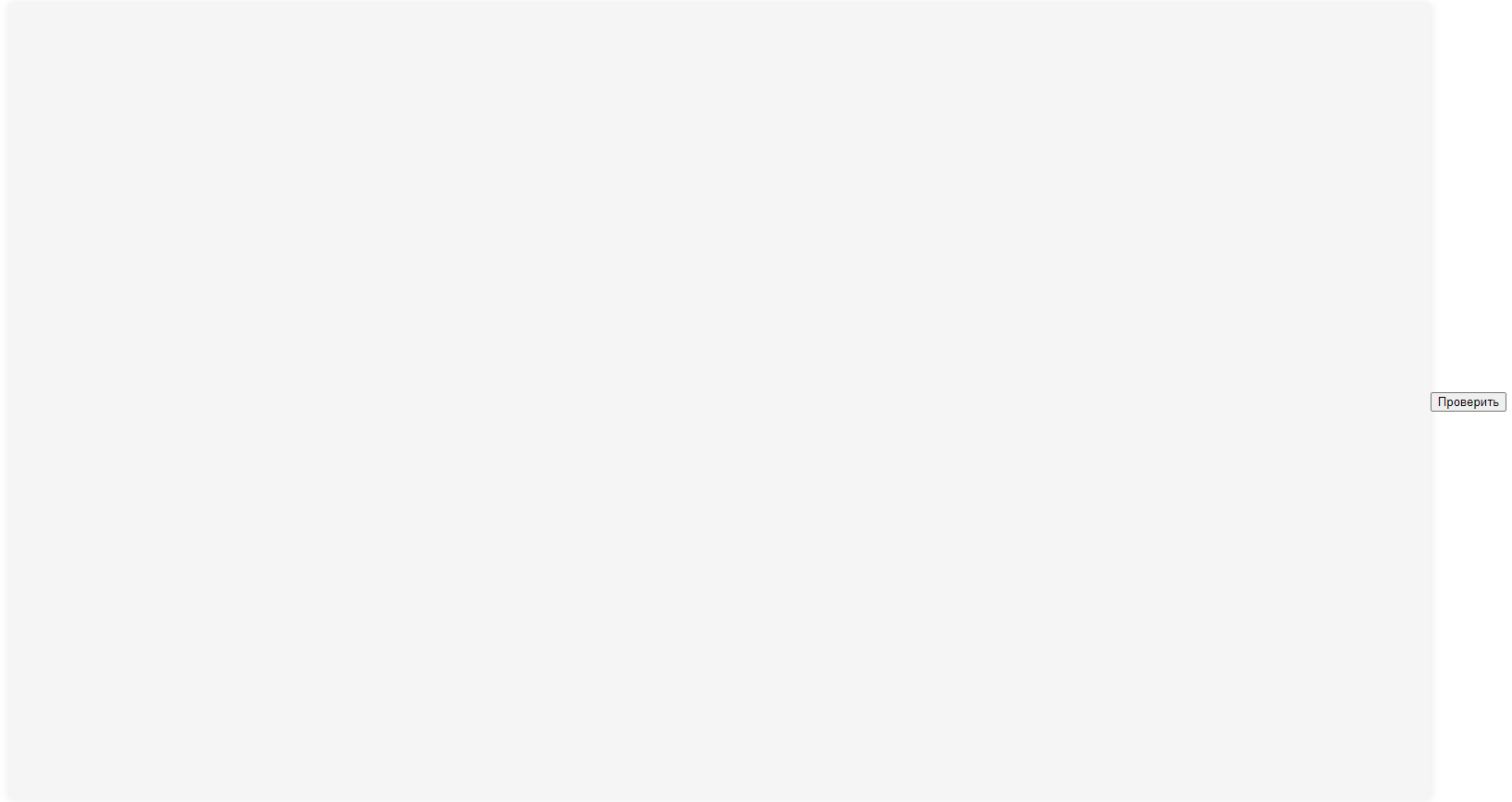


Рисунок 4 – Поле ввода.

Пользователь может написать текст, который подсветит его пунктуационные ошибки (см. рис. 5).

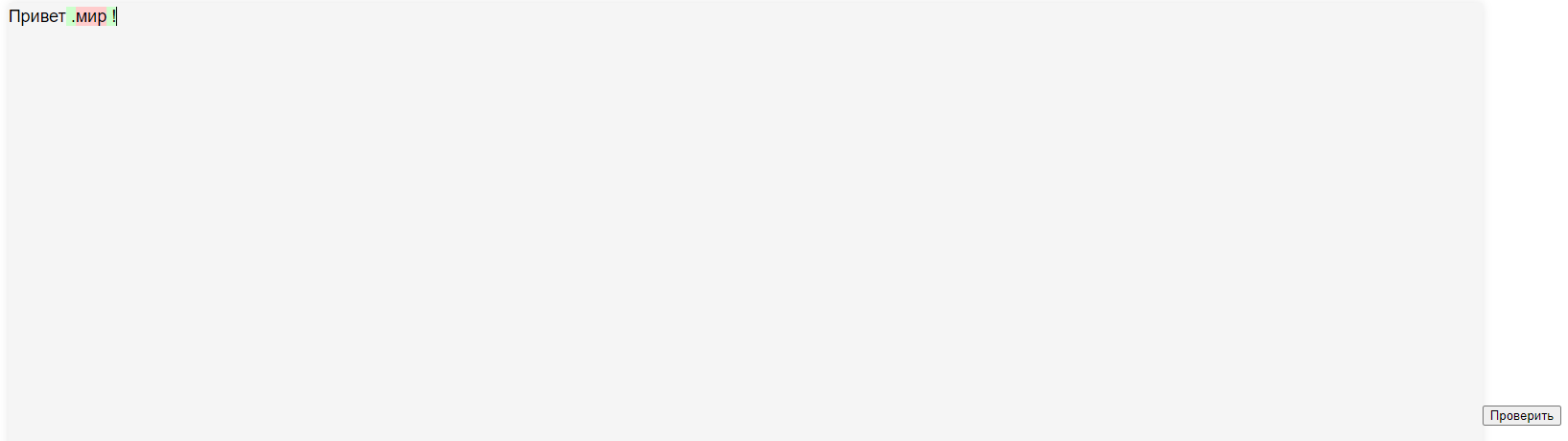


Рисунок 5 – Пример подсветки пунктуационных ошибок.

После ввода текста, пользователь может нажать на кнопку «проверить» и программа подсветит орфографические ошибки, если они есть.

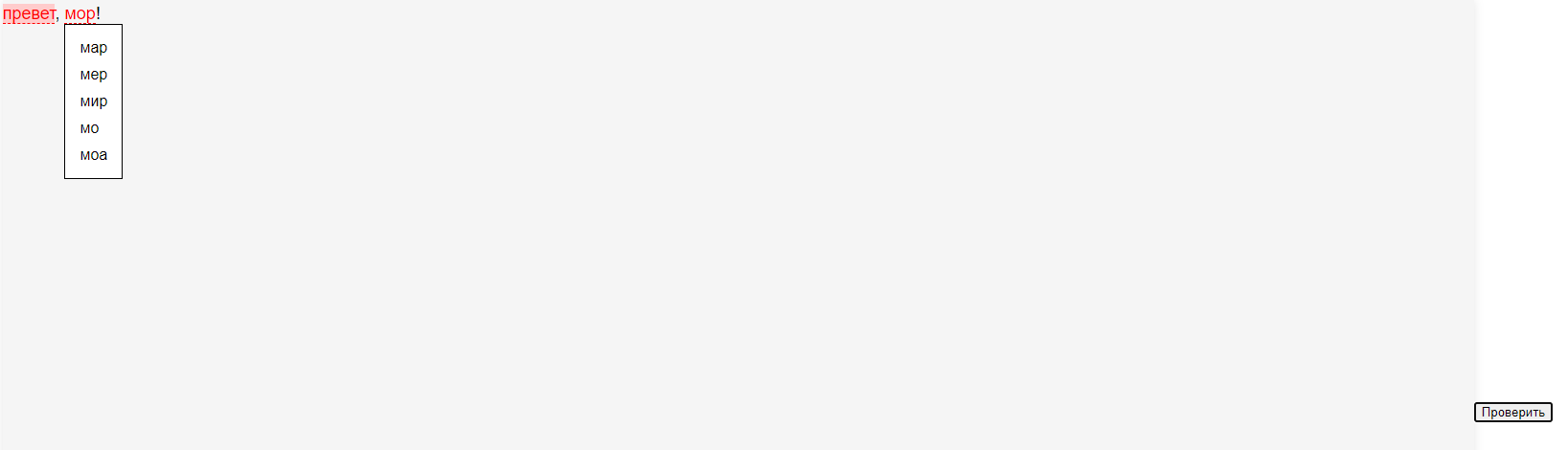


Рисунок 6 – Пример подсветки орфографических ошибок.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были получены навыки разработки веб-приложений с использованием фреймворка Flask, а также интеграции различных технологий для реализации сложной логики. Произведен поиск информации и изучены материалы по предметной области, связанные с алгоритмом расстояния Левенштейна, регулярными выражениями и их применением в автоматической обработке текста.

В результате работы было создано веб-приложение для автоисправления текста. Приложение использует JavaScript для обработки текста с применением регулярного языка и Python для вычисления расстояния Левенштейна. Логика приложения реализована через взаимодействие двух конечных автоматов: один, написанный на JavaScript, выполняет проверку текста с помощью регулярных выражений, а второй, реализованный на Python, отвечает за исправление ошибок. Пользовательский интерфейс позволяет вводить текст, выполнять его проверку и получать варианты исправлений, автоматизируя процесс исправления орфографических и пунктуационных ошибок.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хабр. Расстояние Левенштейна [Электронный ресурс] – Режим доступа:

<https://habr.com/ru/articles/676858/> свободный

1. Хабр, Регулярные выражения (regexp) — основы [Электронный ресурс] - Режим доступа: <https://habr.com/ru/articles/545150/> свободный
2. Хабр. Конечные автоматы в реальной жизни: где мы их используем и почему [Электронный ресурс] - Режим доступа: <https://habr.com/ru/companies/yandex_praktikum/articles/564800/> свободный

# Приложение А

from flask import Flask, request, jsonify

from flask\_cors import CORS

from bisect import bisect\_left

from spellchecker import SpellChecker

app = Flask(\_\_name\_\_)

CORS(app)  # Включаем CORS для работы с фронтендом

# Инициализация орфографического словаря

spell = SpellChecker(language="ru")

filepath = "sorted\_words.txt"  # Файл для пользовательского словаря

# Функции из первого кода

class LevenshteinAutomaton:

    def \_\_init\_\_(self, word, max\_distance):

        self.word = word

        self.max\_distance = max\_distance

    def build\_automaton(self):

        word\_len = len(self.word)

        states = {}

        for i in range(word\_len + 1):

            for j in range(self.max\_distance + 1):

                states[(i, j)] = set()

        for i in range(word\_len + 1):

            for j in range(self.max\_distance + 1):

                if j < self.max\_distance:

                    states[(i, j)].add((i, j + 1))  # Добавление символа

                if i < word\_len:

                    states[(i, j)].add((i + 1, j + 1))  # Удаление символа

                    if j < self.max\_distance:

                        states[(i, j)].add((i + 1, j + 1))  # Замена символа

                    states[(i, j)].add((i + 1, j))  # Совпадение символа

        self.states = states

    def match(self, input\_word):

        current\_states = {(0, 0)}

        for char in input\_word:

            next\_states = set()

            for state in current\_states:

                i, j = state

                if j < self.max\_distance:

                    next\_states.add((i, j + 1))

                if i < len(self.word):

                    next\_states.add((i + 1, j + 1))

                    if self.word[i] == char:

                        next\_states.add((i + 1, j))

                    else:

                        next\_states.add((i + 1, j + 1))

            current\_states = next\_states

        return any(state[0] == len(self.word) and state[1] <= self.max\_distance for state in current\_states)

def load\_dictionary(filepath):

    try:

        with open(filepath, "r", encoding="windows-1251") as file:

            return file.read().splitlines()

    except FileNotFoundError:

        print("Файл словаря не найден. Создайте файл sorted\_words.txt.")

        return []

def suggest\_word(input\_word, dictionary, max\_distance):

    suggestions = []

    start\_index = bisect\_left(dictionary, input\_word[:1])

    for word in dictionary[start\_index:]:

        if not word.startswith(input\_word[:1]):

            break

        lev\_automaton = LevenshteinAutomaton(word, max\_distance)

        lev\_automaton.build\_automaton()

        if lev\_automaton.match(input\_word):

            suggestions.append(word)

        if len(suggestions) >= 5:  # Ограничиваем количество предложений

            break

    return suggestions

# Flask-обработчик

@app.route('/receive-words', methods=['POST'])

def receive\_words():

    data = request.get\_json()

    if not data or 'words' not in data:

        return jsonify({"status": "error", "message": "Invalid data"}), 400

    words = data['words']  # Получаем словарь слов от клиента

    dictionary = load\_dictionary(filepath)  # Загружаем словарь

    max\_distance = 1  # Максимальное расстояние Левенштейна

    suggestions = {}

    for word\_id, word in words.items():

        if word not in spell:  # Если слово некорректное

            suggested\_words = suggest\_word(word, dictionary, max\_distance)

            if suggested\_words:

                suggestions[word\_id] = suggested\_words

    if suggestions:

        response = {

            "status": "error",

            "message": "Words processed",

            "suggestions": suggestions,

        }

    else:

        response = {

            "status": "correct",

            "message": "Words processed",

            "suggestions": suggestions,

        }

    return jsonify(response)

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True, port=5000)

# Приложение Б

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Автоисправление</title>

  <style>

    body {

      margin: 0;

      padding: 0;

      font-family: Arial, sans-serif;

      background-color: #fff;

      display: flex;

      justify-content: center;

      align-items: center;

      height: 100vh;

    }

    #editor {

      width: 80%;

      height: 90%;

      border: none;

      outline: none;

      font-size: 18px;

      line-height: 1.6;

      white-space: pre-wrap;

      word-wrap: break-word;

      background-color: #f5f5f5;

      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

    }

    .error {

      background-color: #ffcccc;

    }

    .tooltip {

      position: absolute;

      background: #f8f8f8;

      border: 1px solid #ccc;

      padding: 5px 10px;

      border-radius: 4px;

      font-size: 12px;

      display: none;

      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);

    }

    .highlight-green {

      background-color: #ccffcc;

    }

  </style>

</head>

<body>

  <div id="editor" contenteditable="true" spellcheck="false"></div>

  <div id="tooltip" class="tooltip"></div>

  <button id="checkWords">Проверить</button>

  <div id="popup" style="display: none; position: absolute; background: white; border: 1px solid black; padding: 10px; z-index: 100;"></div>

  <script>

    const editor = document.getElementById("editor");

    const tooltip = document.getElementById("tooltip");

    editor.addEventListener("input", () => {

      const cursorPosition = saveCursorPosition(editor);

      const text = editor.innerText;

      wrapWordsInSpans(text);

      const spans = collectSpans();

      const errors = checkTextWithFiniteAutomaton(spans);

      restoreCursorPosition(editor, cursorPosition);

    });

    editor.addEventListener("mouseover", (event) => {

      if (event.target.classList.contains("error")) {

        tooltip.innerText = event.target.getAttribute("data-error");

        tooltip.style.display = "block";

        tooltip.style.top = `${event.pageY + 5}px`;

        tooltip.style.left = `${event.pageX + 5}px`;

      }

    });

    editor.addEventListener("mouseout", () => {

      tooltip.style.display = "none";

    });

    function wrapWordsInSpans(text) {

      const wordsAndPunctuation = text.split(/(\s\*[.,!?]\s\*|\s+)/).filter(Boolean);

      let wrappedText = "";

      let wordCounter = 0;

      let spanCounter = 0;

      wordsAndPunctuation.forEach((part) => {

        if (/^\s\*[.,!?]\s\*$/.test(part) || /^\s+$/.test(part)) {

          const spaceId = `space\_${spanCounter++}`;

          wrappedText += `<span id="${spaceId}">${part}</span>`;

        } else if (part.trim().length > 0) {

          const wordId = `word\_${wordCounter++}`;

          wrappedText += `<span id="${wordId}">${part}</span>`;

        }

      });

      editor.innerHTML = wrappedText;

    }

    function collectSpans() {

      const spanElements = document.querySelectorAll('span');

      const spans = Array.from(spanElements).map(span => ({

        id: span.id,

        text: span.textContent,

        type: span.getAttribute('data-error') ? 'error' : 'normal'

      }));

      return spans;

    }

function checkTextWithFiniteAutomaton(spans) {

  const states = {

    START: 0,

    WORD: 1,

    PUNCTUATION: 2,

    SPACE: 3

  };

  let currentState = states.START;

  let isFirstWord = true;

  let consecutiveSpaces = 0;

  let lastSpaceId = null;

  spans.forEach((span, index) => {

    const nextSpan = spans[index + 1];

    const text = span.text;

    switch (currentState) {

      case states.START:

      case states.PUNCTUATION:

        if (span.id.startsWith('word\_')) {

          if (isFirstWord && !/^[A-ZА-Я]/.test(text)) {

            document.getElementById(span.id).classList.add('error');

            document.getElementById(span.id).setAttribute('data-error', 'Предложение должно начинаться с заглавной буквы');

          }

          isFirstWord = false;

          currentState = states.WORD;

          consecutiveSpaces = 0;

        }

        console.log(currentState);

        break;

      case states.WORD:

        if (span.id.startsWith('space\_')) {

          if (/\s+[.,!?]/.test(text)) {

            document.getElementById(span.id).classList.add('highlight-green');

            document.getElementById(span.id).setAttribute('data-error', 'Лишний пробел перед знаком препинания');

          }

          if (text.trim() === '') {

            consecutiveSpaces++;

            lastSpaceId = span.id;

            if (consecutiveSpaces > 1) {

              document.getElementById(span.id).classList.add('highlight-green');

              document.getElementById(span.id).setAttribute('data-error', 'Несколько пробелов подряд');

            }

          }

          if (/[.!?]/.test(text)) {

            isFirstWord = true;

            currentState = states.PUNCTUATION;

          } else {

            currentState = states.SPACE;

          }

        }

        console.log(currentState);

        break;

      case states.SPACE:

        if (span.id.startsWith('space\_')) {

          const prevSpan = span.id;

          console.log(span.id);

          if (/[.,!?]/.test(prevSpan)) {

            document.getElementById(span.id).classList.add('highlight-green');

            document.getElementById(span.id).setAttribute('data-error', 'Отсутствует пробел после знака препинания');

          }

          currentState = states.WORD;

          consecutiveSpaces = 0;

        }

        console.log(currentState);

        break;

    }

  });

}

    function saveCursorPosition(element) {

      const selection = window.getSelection();

      if (!selection.rangeCount) return null;

      const range = selection.getRangeAt(0);

      const preRange = range.cloneRange();

      preRange.selectNodeContents(element);

      preRange.setEnd(range.startContainer, range.startOffset);

      const position = preRange.toString().length;

      return position;

    }

    function restoreCursorPosition(element, position) {

      if (position === null) return;

      const selection = window.getSelection();

      const range = document.createRange();

      let charIndex = 0;

      let found = false;

      function traverseNodes(node) {

        if (found) return;

        if (node.nodeType === Node.TEXT\_NODE) {

          const nextCharIndex = charIndex + node.length;

          if (position >= charIndex && position <= nextCharIndex) {

            range.setStart(node, position - charIndex);

            range.collapse(true);

            found = true;

          }

          charIndex = nextCharIndex;

        } else {

          node.childNodes.forEach(traverseNodes);

        }

      }

      traverseNodes(element);

      selection.removeAllRanges();

      selection.addRange(range);

    function collectWords() {

      const spans = document.querySelectorAll('span[id^="word\_"]');

      const wordsDictionary = {};

      spans.forEach((span) => {

        const id = span.id;

        const word = span.textContent.trim();

        wordsDictionary[id] = word;

      });

    }

// Пример вызова функции для проверки

collectWords();

    }

// Функция отправки данных на сервер

async function sendWordsToServer(wordsDictionary) {

  try {

    const response = await fetch('http://localhost:5000/receive-words', {

      method: 'POST',

      headers: {

        'Content-Type': 'application/json',

      },

      body: JSON.stringify({ words: wordsDictionary }),

    });

    const result = await response.json();

    console.log('Response from server:', result);

    processServerResponse(result);

  } catch (error) {

    console.error('Error sending data to server:', error);

  }

}

// Функция сбора слов со страницы и отправки на сервер

function collectWordsAndSend() {

  const spans = document.querySelectorAll('span[id^="word\_"]');

  const wordsDictionary = {};

  spans.forEach((span) => {

    const id = span.id;

    const word = span.textContent.trim();

    wordsDictionary[id] = word;

  });

  console.log('Sending words:', wordsDictionary);

  sendWordsToServer(wordsDictionary);

}

// Обработка ответа от сервера

function processServerResponse(response) {

  if (response.status === "error") {

    const suggestions = response.suggestions;

    for (const wordId in suggestions) {

      const span = document.getElementById(wordId);

      if (span) {

        span.style.color = "red";

        span.style.borderBottom = "1px dashed red";

        // Добавляем обработчик для показа pop-up

        span.addEventListener("mouseover", (event) => {

          showPopup(event, span, suggestions[wordId]);

        });

      }

    }

  }

}

// Функция показа pop-up

function showPopup(event, span, suggestions) {

  const popup = document.getElementById("popup");

  // Очищаем содержимое pop-up

  popup.innerHTML = "";

  // Добавляем варианты замены

  suggestions.forEach((suggestion) => {

    const suggestionElement = document.createElement("div");

    suggestionElement.textContent = suggestion;

    suggestionElement.style.cursor = "pointer";

    suggestionElement.style.padding = "5px";

    suggestionElement.addEventListener("click", () => {

      span.textContent = suggestion; // Меняем текст в span

      hidePopup(); // Скрываем pop-up

      span.style.color = ""; // Убираем подсветку

      span.style.borderBottom = "";

    });

    popup.appendChild(suggestionElement);

  });

  // Позиционируем и показываем pop-up

  const rect = span.getBoundingClientRect();

  popup.style.left = `${rect.left + window.scrollX}px`;

  popup.style.top = `${rect.bottom + window.scrollY}px`;

  popup.style.display = "block";

  // Добавляем глобальный обработчик для клика вне pop-up

  document.addEventListener("click", outsideClickListener);

}

// Функция скрытия pop-up

function hidePopup() {

  const popup = document.getElementById("popup");

  popup.style.display = "none";

  // Удаляем обработчик клика вне pop-up

  document.removeEventListener("click", outsideClickListener);

}

// Обработчик клика вне pop-up

function outsideClickListener(event) {

  const popup = document.getElementById("popup");

  if (!popup.contains(event.target)) {

    hidePopup();

  }

}

// Обработчик кнопки для проверки слов

document.getElementById("checkWords").addEventListener("click", collectWordsAndSend);

  </script>

</body>

</html>