

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГУ»)**

Факультет компьютерных наук

Кафедра информационных систем

Реализация токенизации, лемматизации и стемминга текста с помощью

Python-библиотеки rymorphy2

Отчет по учебной практике, ознакомительной
09.03.02 Информационные системы и технологии
4 семестр 2021/2022 учебного года

Зав. кафедрой



Борисов Д. Н., к.т.н., доцент

Обучающийся



Кудинов И. М.

Руководитель



Сычев А. В., к.ф.-м.н., доцент

Воронеж 2022

СОДЕРЖАНИЕ

1 Постановка задачи	3
1.1 Введение	3
1.2 Ожидаемый функционал	3
1.3 Задачи.....	4
2 Методы решения задачи	5
3 Этапы решения задачи.....	5
3.1 Алгоритм токенизации текста.	5
3.2 Алгоритм обработки текста	7
3.3 Подготовка данных для корректного отображения пользователю.....	8
3.4 Создание пользовательского интерфейса	10
3.5 Получение данных с формы стартовой страницы	14
3.6 Обработка URL-запросов приложения	15
3.7 Загрузка приложения на удаленный сервер	16
4 Инструментальные средства разработки.....	18
4.1 PyCharm	18
4.2 HTML	18
4.3 CSS.....	18
4.4 Bootstrap.....	19
4.5 Flask.....	19
4.6 Pymorphy2	20
4.7 Gunicorn.....	20
4.8 Heroku.....	20
5 Описание структуры разработанного приложения.	20
6 Описание функционала и интерфейса приложения (со скриншотами и комментариями)	22
7 Задачи, выполненные в ходе разработки проекта.	24
9 Оценка степени завершенности и перспективы развития проекта.	26

1 Постановка задачи

1.1 Введение

Обработка естественного языка сейчас не используются разве что в совсем консервативных отраслях. В большинстве технологических решений распознавание и обработка «человеческих» языков давно внедрена: именно поэтому обычный IVR с жестко заданными опциями ответов постепенно уходит в прошлое, чатботы начинают все адекватнее общаться без участия живого оператора, фильтры в почте работают на ура и т.д. Именно для реализации данных. Алгоритмы машинного обучения не могут напрямую работать с сырым текстом, поэтому необходимо конвертировать текст в наборы цифр (векторы). Это называется извлечением признаков.

Токеназация, лематизация и стемминг текста являются подходами обработки естественного языка. Natural Language Processing (далее – NLP) – обработка естественного языка – подраздел информатики и искусственного интеллекта, посвященный тому, как компьютеры анализируют естественные (человеческие) языки. NLP позволяет применять алгоритмы машинного обучения для текста и речи. Например, мы можем использовать NLP, чтобы создавать системы вроде распознавания речи, обобщения документов, машинного перевода, выявления спама, распознавания именованных сущностей, ответов на вопросы, автоисправления, предиктивного ввода текста и т.д. Обработка естественного языка является основой голосовых помощников

1.2 Ожидаемый функционал

Разрабатываемое приложение должно иметь веб-интерфейс с возможностью загрузки текста (из файла или вручную). Полученный текст должен токенизироваться по словам, далее слова приводиться к начальным формам и подсчитываться количество вхождения каждой леммы в текст. Приведение слова к начальной форме достигается как путём стемминга, так и

лемматизацией. Стемминг использует в основе своей работы отбрасывание окончания слова. Из этого исходит определяющий минус данного подхода – огромное количество ошибок в начальных формах слов. Лемматизация использует словари и на основе лемм определяет начальную форму слова. Поэтому в разрабатываемом приложении будет применяться алгоритм лемматизации, в силу своего превосходства над стеммингом. Так же необходимо для каждой леммы определять часть речи. После чего отображать полученные данные в виде таблицы на динамически генерируемой HTML-странице.

1.3 Задачи

- Написать алгоритм токенизации входного текста. Как по предложениям, так и по словам
- Написать алгоритм обработки текста, который включает в себя: токенизацию исходного текста по словам, лемматизацию токенов и составления мешка слов, так же алгоритм должен определять часть речи для каждой леммы
- Написать пользовательский интерфейс, который включает в себя: стартовую страницу, с возможностью загрузки текстового файла или написания текста в специальное поле, страницу с информацией об авторе и назначении сайта, а также страницу, на которой будет отображаться результат обработки текста в виде таблицы
- Загрузить приложение на хостинг

2 Методы решения задачи

Приложение будет написано в среде разработки PyCharm — интегрированная среда разработки для языка программирования Python. Так как приложение подразумевает браузерный web-интерфейс, в нем будут использоваться URL-адреса и HTTP-методы (GET и POST). Для более удобной разработки пользовательского приложения будет использован Python фреймворк Flask. Flask — фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Он обеспечивает основные возможности маршрутизации URL-адресов и визуализации страниц. Для морфологического анализа текста будет использована библиотека `ru morphology`. Для корректной работы приложения на удаленном сервере понадобится библиотека `gunicorn`. Для загрузки приложения на хостинг будет использован сервис Heroku.

3 Этапы решения задачи

3.1 Алгоритм токенизации текста.

Т.к в библиотеке `ru morphology` не предусмотрено токенизирование текста, необходимо реализовать собственный алгоритм токенизации исходного текста. В качестве токенов будут использоваться слова, но также будет реализована токенизация по предложениям.

Требования от алгоритма: алгоритм должен разбивать исходную строку на подстроки по пробелам. Далее алгоритму необходимо убрать все знаки препинания, кроме сокращений слов - например: к.т.н. — кандидат технических наук или т.е. — то есть. В остальных словах любые знаки необходимо убрать (Рисунок – 3). Так же алгоритм должен убрать оставшиеся после этого пустые токены, чтобы не засорять входные данные (Рисунок – 1).

Пример. Входная строка: к.т.н. — кандидат технических наук. После

токенизации получается следующий массив: ['к.т.н.', '—', 'кандидат', 'технических', 'наук.']. После удаления лишних знаков: ['к.т.н.', '', 'кандидат', 'технических', 'наук']. Необходимо удалить пустой токен ''.

Для токенизации по предложениям необходимо последовательно перебирать все символы исходного текста. Если текущий символ – знак препинания конца предложения, добавлять предложение в массив.

Применить этот алгоритм к каждому предложению (Рисунок – 2).

Ниже будет представлен код функций, реализующих данные алгоритмы с пояснениями.

```
def word_tokenize(text: str):
    words = text.split(' ') # разделение на слова по пробелам
    words = remove_patterns(words) # функция для удаления ненужных символов (чисел, знаков препинания букв английского алфавита)
    words = remove_empty_values_from_list(words, ['', '.']) # функция для удаления пустых значений из токенов
    return words

def remove_empty_values_from_list(input_list, values):
    for value in values:
        input_list = list(filter(value.__ne__, input_list))
    return input_list

def remove_patterns(words: list):
    words = remove_empty_values_from_list(words, [''])
    for index in range(len(words)):
        word = words[index]
        if word[-1] == '.':
            if index == len(words) - 1:
                word = remove_char(word, len(word) - 1)
            else:
                if words[index + 1][0].isupper():
                    word = remove_char(word, len(word) - 1)
        for char in word:
            if char in patterns:
                word = word.replace(char, '')
        words[index] = word
    return words
```

Рисунок 1 - Функция токенизации по словам

«word_tokenize» - функция для токенизации по словам

«remove_empty_values_from_list» - функция для удаления передаваемых значений из списка

«remove_patterns» - функция для удаления передаваемых символов из элементов листа

```
def sentence_tokenize(text: str):
    sentences = [] # массив для предложений
    current_sentence = '' # строка для записи текущего предложения посимвольно
    for index in range(len(text)): # цикл для перебора каждого символа исходного текста
        char = text[index]
        if char == '.' or char == '!' or char == '?' and index != len(text) - 1:
            if check_upper_case(text[index + 1:]):
                sentences.append(current_sentence) # если встречается знак препинания для конца предложения
                # и следующее слово начинается с заглавной буквы - помещаем текущее предложение в массив с предложениями
                # и обнуляем строку с текущим предложением
                current_sentence = ''
            else:
                current_sentence = current_sentence + char
        else:
            if char == ' ' and len(current_sentence) == 0:
                continue
            else:
                current_sentence = current_sentence + char

    sentences.append(current_sentence)
    return sentences
```

Рисунок 2 - Функция токенизации по предложениям

```
patterns = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#%$;%:&^*\"'() ,/<>{}[]\\|_-=+`~!@#%$;%:&^*\"'() ,/<>{}[]\\|_-=+`~"
```

Рисунок 3 - Patterns

3.2 Алгоритм обработки текста

Он включает в себя: токенизацию исходного текста по словам, лемматизацию токенов и составления мешка слов, так же алгоритм должен определять часть речи для каждой леммы.

В библиотеке `rumorphy2` есть метод `parse`. Он предоставляет всю информацию по переданному слову. Необходимо к каждому токenu применить данный метод, после чего получить информацию о начальной форме слова и её части речи, а также обновить счетчик встречаемости данной леммы в тексте. Для решения задачи подсчета количества вхождений леммы в текст необходимо использовать структура данных – словарь. Т.к он обеспечивает наиболее удобный функционал для данной задачи. Полученная информация помещается в словарь, где ключ – начальная форма слова, а значение – кортеж [количество вхождений в текст, часть речи]. Как только все токены будут, мы получим словарь с данными и посчитанными

значениями (Рисунок - 6).

Для корректного и удобного хранения данных создадим класс Word, который будет содержать в себе три поля: начальная форма, часть речи и количество вхождений в текст (Рисунок – 4). При переборе каждого ключа словаря будет создаваться новый объект Word и помещаться в массив.

3.3 Подготовка данных для корректного отображения пользователю

В библиотеке rymorphy2 часть речи обозначается сокращениями на английском языке (ADJF - Adjective name (full)). Данный формат записи вводит пользователя в заблуждение. Необходимо реализовать функцию, которая заменяла бы сокращения на полные названия частей речи для удобного отображения. Создадим класс перечисления (Enum), в котором каждому сокращению сопоставим в качестве значения полное название этой части речи (Рисунок - 5). Далее функция просто найдет совпадение, перебирая все сокращения, и вернет его значение. Если какое-то значение не было обнаружено, функция вернет строку «Not defined» (Рисунок – 6)

```
class Word:
    def __init__(self, normal_form: str, part_of_speech: str, number_of_entries: int):
        self.normal_form = normal_form
        self.part_of_speech = part_of_speech
        self.number_of_entries = number_of_entries

    def __str__(self):
        return self.normal_form + '|' + self.part_of_speech + '|' + str(self.number_of_entries)

    def __repr__(self):
        return repr((self.normal_form, self.part_of_speech, self.number_of_entries))
```

Рисунок 4 - Класс Word


```

class Part_of_speech(Enum):
    NOUN = "Noun"
    ADJF = "Adjective name (full)"
    ADJS = "Adjective name (short)"
    COMP = "Comparative"
    VERB = "Verb"
    INFN = "Verb"
    PRTF = "Participle (full)"
    PRTS = "Participle (short)"
    GRND = "Gerund"
    NUMR = "Numeral"
    ADVB = "Adverb"
    NPRO = "Pronoun"
    PRED = "Predicative"
    PREP = "Preposition"
    CONJ = "Conjunction"
    PRCL = "Particle"
    INTJ = "Interjection"

```

Рисунок 5 - Класс перечисления частей речи

Рисунок 6 -

```

def get_part_of_speech(part_of_speech):
    if part_of_speech is None:
        return "Not defined"
    for current_part in Part_of_speech:
        if current_part.name == part_of_speech:
            return current_part.value
    return None

```

Рисунок 7 - Функция для получения полного названия части речи

```

def text_process(text: str):
    tokens = Tokenization.word_tokenize(text) # токенизация текста
    morph = pymorphy2.MorphAnalyzer()
    tags = {} # словарь для значений
    words = [] # массив для выходных данных

    for token in tokens: # перебор всех токенов и добавлений полученных данных в словарь
        word = morph.parse(token)[0]
        normal_form = word.normal_form
        part_of_speech = get_part_of_speech(word.tag.POS)

        if normal_form not in tags.keys():
            tags[normal_form] = [1, part_of_speech]
        else:
            tags[normal_form][0] = tags[normal_form][0] + 1

    for key in tags.keys(): # создание объектов по ключам словаря
        words.append(Word(key, tags[key][1], tags[key][0]))

    return words

```

Рисунок 8 - Функция обработки текста

3.4 Создание пользовательского интерфейса

В качестве пользовательского интерфейса будет использоваться web-страница. Для реализации этой задачи будет использован язык разметки HTML, формальный язык описания внешнего вида web-страницы CSS, фреймворк Flask, свободный набор инструментов для создания сайтов и веб-приложений Bootstrap. Фреймворк Flask позволяет наследовать html файлы. Это позволит сделать удобную структуру файлов разметки.

На базовой странице необходимо создать Header страницы, Footer страницы, общий фон сайта. Header содержит кнопку возврата на стартовую и кнопку перехода на информационную страницу. Footer содержит ссылки на GitHub и Email. А затем все страницы интерфейса наследовать от базовой, динамически подставляя необходимый код разметки страницы (Рисунок – 8).

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-DevHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL60i01tfPri4tjfhXaWutUpFmBp4vmVor" crossorigin="anonymous">
  <link rel="stylesheet" href="/static/style.css">
  <title>{% block title %}{% endblock %}</title>
  <link rel="icon" href="/static/images/icon.svg">
</head>
<body>
<div class="wrapper">
  <header class="header">
    <div class="container">
      <div class="header__top">
        <a href="/" class="header__logo"></a>
        <a href="/" class="header__title">Natural Language Processing</a>
        <div class="header__inner">
          <a href="/about" class="header__about">About</a>
        </div>
      </div>
    </div>
  </header>

  <div class="container">
    {% block content %}{% endblock %}
  </div>

  <footer class="footer">
    <div class="container">
      <div class="copyright__wrapper">
        <div class="copyright">
          © 2022 NLP by Ivan Kudinov
        </div>
        <div class="footer__link">
          <a href="https://github.com/joolsoul" target="_blank" class="footer__logo"></a>
          <a href="https://e.mail.ru/compose/" target="_blank" class="footer__logo"></a>
        </div>
      </div>
    </div>
  </footer>
</div>
</body>
</html>

```

Рисунок 9 - Base_page.html

На стартовой странице необходимо реализовать форму для отправки post-запроса. Форма должна состоять из поля для ввода текста с клавиатуры, а также форму для загрузки файлов с разрешением .doc .docx .txt. Данная страница наследуется от базовой и содержит в себе основной Header и Footer (Рисунок – 9).

```
{% extends "base_page.html" %}
{% block title %}
    Start page
{% endblock %}
{% block content %}
    <main class="main">
        <div class="container">
            <div class="ribbon"></div>
            <div class="input__form">
                <h1>Enter the text to be processed or upload a text file</h1>
                <form action="/result" method="POST" enctype="multipart/form-data">
                    <div class="block__input">
                        <textarea class="form-textarea" name="input_textarea" placeholder="Input text"></textarea>
                    </div>
                    <div class="block__input">
                        <input class="form-input" name="input_file" type="file" accept=".doc,.docx,.txt">
                    </div>
                    <button>Process text</button>
                </form>
            </div>
        </div>
    </main>
{% endblock %}
```

Рисунок 10 - Start_page.html

На информационной странице необходимо написать информацию об авторе и назначении сайта, а также контакты для связи. Данная страница также унаследована от базовой и имеет все её элементы (Рисунок – 10).

```

{% extends "base_page.html" %}
{% block title %}
    About
{% endblock %}
{% block content %}

    <div class="main__about__text">
        This project was created as part of an introductory summer practice. <br>
        <span class="my__font">Author:</span> Ivan Kudinov <br>
        <span class="my__font">Head:</span> Sychev Alexander Vasilyevich <br>
        This site implements natural language processing. For the site to work, download the source file with the text
        or enter it in the appropriate form. After processing the text, you will be presented with a page with the source text,
        as well as a table with vectors. Each vector contains a lemma (the initial form of a word), the part of speech to
        which it relates, as well as the number of occurrences of this lemma in the text. Vectors are sorted by occurrence in
        the text.

        <br>

        <span class="main__about__contacts">If you have any questions , please contact us by email ivan.kudinov.2016@mail.ru</span>

        <br>

        <span class="main__about__after">Sincerely, Kudinov Ivan</span>
    </div>
{% endblock %}

```

Рисунок 11 - About_page.html

На странице отображения результата необходимо реализовать таблицу с обработанной информацией, а также указать исходный текст. Все данные должны подставляться динамически. В решении данной задачи помогает фреймворк Flask и Bootstrap. Flask дает возможность динамической генерации шаблонов html-страницы. Bootstrap позволяет стилизовать таблицу, не используя для этого отдельный css файл со стилями. При генерации страницы передается исходный текст. Далее он подставляется в соответствующую форму на странице. Также передается массив с элементами. В цикле перебирается каждый из элементов, берется информация и подставляется в соответствующую колонку в таблице (Рисунок – 11).

```

{% extends "base_page.html" %}
{% block title %}
    Result page
{% endblock %}
{% block content %}
    {% block text %}
        <div class="input__text">
            Input text:
        </div>
        <div class="main__text">
            {{ text }}
        </div>
    {% endblock %}
    <hr>
    <div class="table__text">
        <table id="data" class="table table-striped">
            <thead>
                <tr>
                    <th>The initial form of the word</th>
                    <th>Part of speech</th>
                    <th>Number of entries in the text</th>
                </tr>
            </thead>
            <tbody>
                {% for word in words %}
                    <tr>
                        <td>{{ word.normal_form }}</td>
                        <td>{{ word.part_of_speech }}</td>
                        <td>{{ word.number_of_entries }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
{% endblock %}

```

Рисунок 12 - Result_page.html

3.5 Получение данных с формы стартовой страницы

Алгоритм получения данных должен считывать данные с текстового поля и обрабатывать файл. Входные данные будут записываться в текстовый файл внутри приложения. Перед получением данных, функция должна очистить данный буферный файл, чтобы избежать обработки предыдущего текста. Далее необходимо определить в какой из форм содержатся входные данные. Если текстовое поле содержит текст, а также загружен файл, необходимо записать данные из обоих источников (Рисунок -12)

```

def check_input():
    FileUtils.write_to_file("input", " ", "w")
    if request.form.get("input_textarea") == '':
        uploaded_file = request.files['input_file']
        if uploaded_file.filename != '':
            uploaded_file.save(os.path.join('static/input'))
    else:
        uploaded_file = request.files['input_file']
        argument = 'w'
        if uploaded_file.filename != '':
            uploaded_file.save(os.path.join('static/input'))
            argument = 'a'
        FileUtils.write_to_file("input", ' ', argument)
    input_text = request.form.get("input_textarea")
    FileUtils.write_to_file("input", input_text, argument)

```

Рисунок 13 - Функция чтения входных данных

3.6 Обработка URL-запросов приложения

Для каждого из запросов необходимо написать функцию обработки. При обработке запроса стартовой страницы необходимо просто сгенерировать html-страницу из файла. При нажатии на кнопку отправки формы происходит переадресация на страницу отображения результата, а также отправка POST-запроса с исходным текстом.

При обработке POST-запроса страницы отображения результата необходимо получить данные из формы стартовой страницы. Далее записать эти данные в буферный файл. После чего приложение считывает информацию из этого буферного файла. Произведет обработку текста, отсортирует полученные данные по количеству вхождений леммы в текст и передаст данные для динамического отображения на html-странице.

При обработке запроса страницы с информацией необходимо просто сгенерировать html-страницу из файла (Рисунок – 13).

```

@app.route('/')
def start():
    return render_template('start_page.html')

@app.route('/result', methods=['GET', 'POST'])
def result():
    if request.method == 'GET':
        return render_template("result_page.html")

    check_input()
    text = FileUtils.read_from_file("input")
    words = text_process(text)
    words = sorted(words, key=lambda word: word.number_of_entries, reverse=True)

    return render_template('result_page.html', text=text, words=words)

@app.route('/about')
def about():
    return render_template("about_page.html")

```

Рисунок 14 - Функции обработки URL-запросов

«start» - функция обработки запроса стартовой страницы

«result» - функция обработки запроса страницы отображения результата

«about» - функция обработки запроса страницы с информацией

3.7 Загрузка приложения на удаленный сервер

Для корректной работы приложения на сервере, необходимо установить библиотеку `gunicorn`. Далее необходимо создать файл `requirements.txt`, где будут содержаться все пакеты, необходимые для работы приложения, а также `Procfile`, где будут прописаны параметры запуска приложения на сервере (Рисунок – 14 и Рисунок – 15).


```
click==8.1.3
colorama==0.4.4
DAWG-Python==0.7.2
docopt==0.6.2
Flask==2.1.2
gunicorn==20.1.0
importlib-metadata==4.11.4
itsdangerous==2.1.2
Jinja2==3.1.2
MarkupSafe==2.1.1
pymorphy2==0.9.1
pymorphy2-dicts-ru==2.4.417127.4579844
Werkzeug==2.1.2
zipp==3.8.0
```

Рисунок 15 - Файл requirements.txt

```
web: gunicorn --bind 0.0.0.0:$PORT app:app
```

Рисунок 16 - Файл Procfile

4 Инструментальные средства разработки

4.1 PyCharm

Для написания приложения использовалась среда разработки PyCharm. PyCharm - это интегрированная среда разработки для Python, которая имеет полный комплект средств, необходимых для эффективного программирования на Python. PyCharm имеет удобный редактор кода со всеми полезными функциями: подсветкой синтаксиса, автоматическим форматированием, дополнением и отступами. PyCharm позволяет проверять версии интерпретатора языка на совместимость, а также использовать шаблоны кода. PyCharm имеет большую коллекцию плагинов и позволяет легко устанавливать любые пакеты и библиотеки. Он так же оснащён встроенным терминалом.

4.2 HTML

Для написания web-интерфейса использовался язык разметки HTML. HTML - стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора. Элементы HTML являются строительными блоками HTML страниц. С помощью HTML разные конструкции, изображения и другие объекты, такие как интерактивная веб-форма, могут быть встроены в отображаемую страницу. HTML предоставляет средства для создания заголовков, абзацев, списков, ссылок, цитат и других элементов.

4.3 CSS

Для описания внешнего вида web-страниц был использован язык CSS. CSS - формальный язык описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки (чаще всего HTML или

XHTML). CSS используется создателями веб-страниц для задания цветов, шрифтов, стилей, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS является ограждение и отделение описания логической структуры веб-страницы (которое производится с помощью HTML или других языков разметки) от описания внешнего вида этой веб-страницы (которое теперь производится с помощью формального языка CSS). Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление, чтение голосом (специальным голосовым браузером или программой чтения с экрана) или при выводе устройствами, использующими шрифт Брайля.

4.4 Bootstrap

Для динамической загрузки стилей была использована технология Bootstrap. Bootstrap - свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения.

4.5 Flask

Для создания web-приложения был использован фреймворк Flask. Flask - это упрощенная платформа Python для веб-приложений, которая обеспечивает основные возможности маршрутизации URL-адресов и визуализации страниц. Flask называют "микро"-платформой, так как она не предоставляет напрямую такие функции, как проверка форм, абстракция базы данных, проверка подлинности и т. д. Эти функции предоставляются специальными пакетами Python, называемыми расширениями Flask.

Расширения легко интегрируются с Flask и отображаются так, как будто являются частью самой среды Flask. Например, Flask не предоставляет модуль шаблонов страницы. Использование шаблонов обеспечивается расширениями, такими как Jinja и Jade, как демонстрируется в этом учебнике.

4.6 Rymorphy2

Для символьной и статистической обработки естественного языка был использован пакет библиотек и программ rymorphy2. Rymorphy2 - морфологический анализатор, разработанный на языке программирования Python. Выполняет лемматизацию и анализ слов, способен осуществлять склонение по заданным грамматическим характеристикам слов.

4.7 Gunicorn

Для корректной работы приложения на удаленном сервере была использована библиотечка gunicorn. Gunicorn - это Application-сервер для запуска Web-приложений написанных на Python. Основная его задача — это работа в режиме демона и поддержка постоянной работы Web-приложений.

4.8 Heroku

Для загрузки приложения на удаленный сервер использовался сервис Heroku. Heroku - облачная PaaS-платформа, поддерживающая ряд языков программирования.

5 Описание структуры разработанного приложения.

В корневой папке проекта находятся следующие файлы и пакеты:

- Пакет static – используется фреймворком Flask. Необходим для размещения статических файлов приложения.
 - Пакет images – содержит изображения для пользовательского интерфейса.

- Пакет test_input – содержит текстовый файл с тестовым текстом для загрузки в приложение и проверки его работоспособности
- Файл input.txt – буферный файл, куда записывается текст после получения из формы на стартовой странице. Из этого файла приложение берет исходные данные.
- Файл style.css – файл стилей для html-страниц
- Пакет templates - используется фреймворком Flask. Необходим для размещения шаблонов html-страниц
 - Файл start_page.html – файл для отображения стартовой страницы
 - Файл about_page.html – файл для отображения страницы с информацией
 - Файл base_page.html – файл с базовой разметкой для всех страниц
 - Файл result_page.html – файл для отображения страницы с результатом
- Пакет util – содержит в себе вспомогательные классы.
 - FileUtils – класс для работы с файлами. Записи в файл и чтения из него
 - TokenizationUtils – класс для токенизации текста. Токенизации по предложениям и по словам.

- Файл app.py – главный исполняемый файл приложения. Содержит в себе основную логику, а также точку входа в программу. Также отвечает за обработку URL-запросов web-интерфейса.
- Файл Procfile – файл конфигурации
- Файл requirements.txt – файл конфигурации

6 Описание функционала и интерфейса приложения (со скриншотами и комментариями)

Стартовая страница имеет Header, который содержит ссылку для возврата на стартовую страницу и ссылку на страницу с информацией (About), и Footer, который содержит ссылки на GitHub и Email (Рисунок – 17). Данная страница позволяет загрузить файл с расширением .doc .docx .txt. Так же есть поле для набора текста с клавиатуры и кнопка отправка формы, которая имеет три состояния (Пассивная (Рисунок – 17), Активная (Рисунок – 18), Нажатая (Рисунок – 19)).

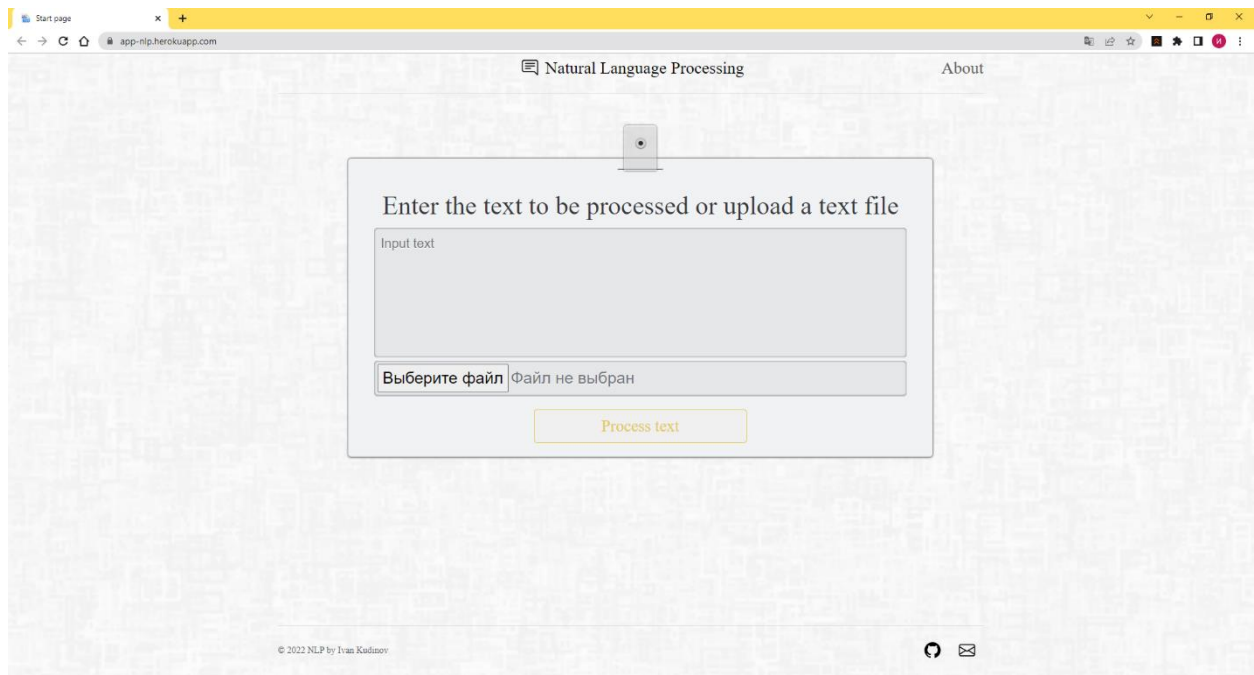


Рисунок 18 - Стартовая страница

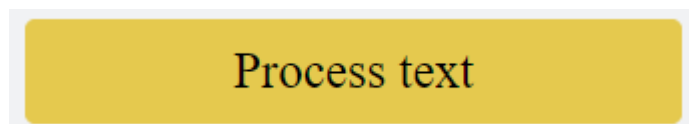


Рисунок 19 - Активная кнопка

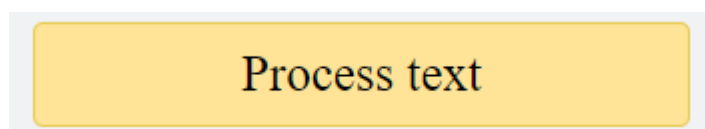


Рисунок 20 - Нажатая кнопка

Страница с информацией содержит текст. Информацию о назначении данного приложения, цели создания, авторе и руководителе, а также контактную информацию (Рисунок – 20).

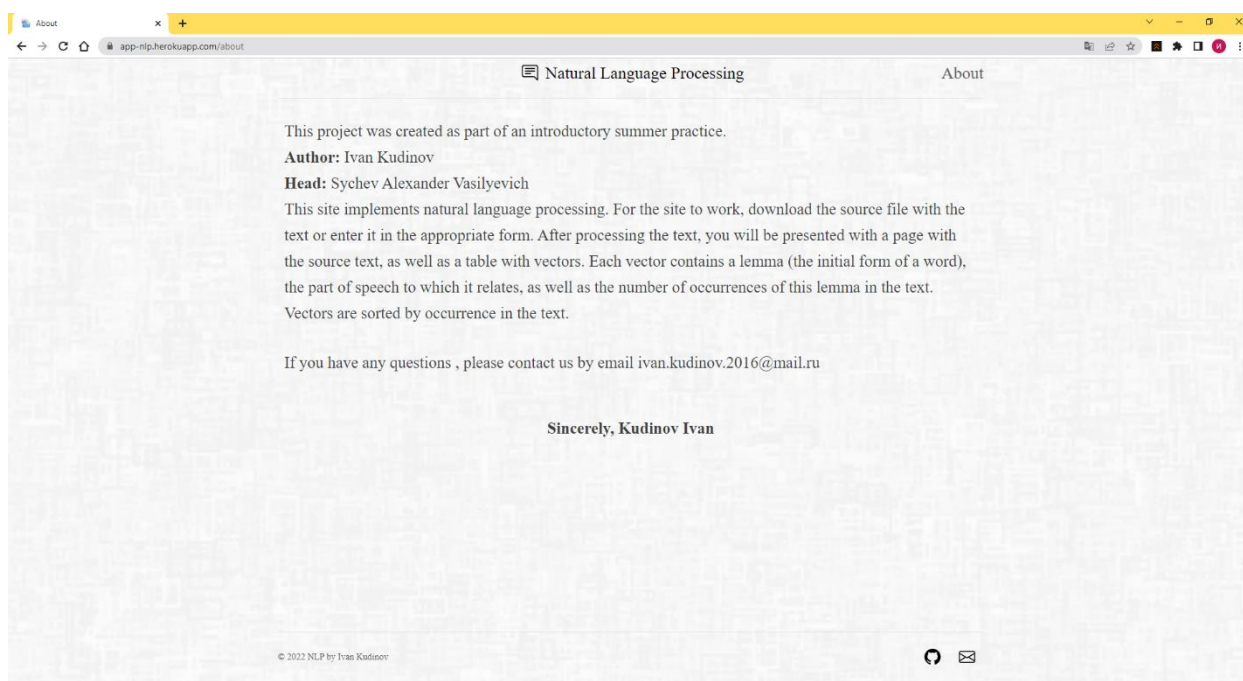
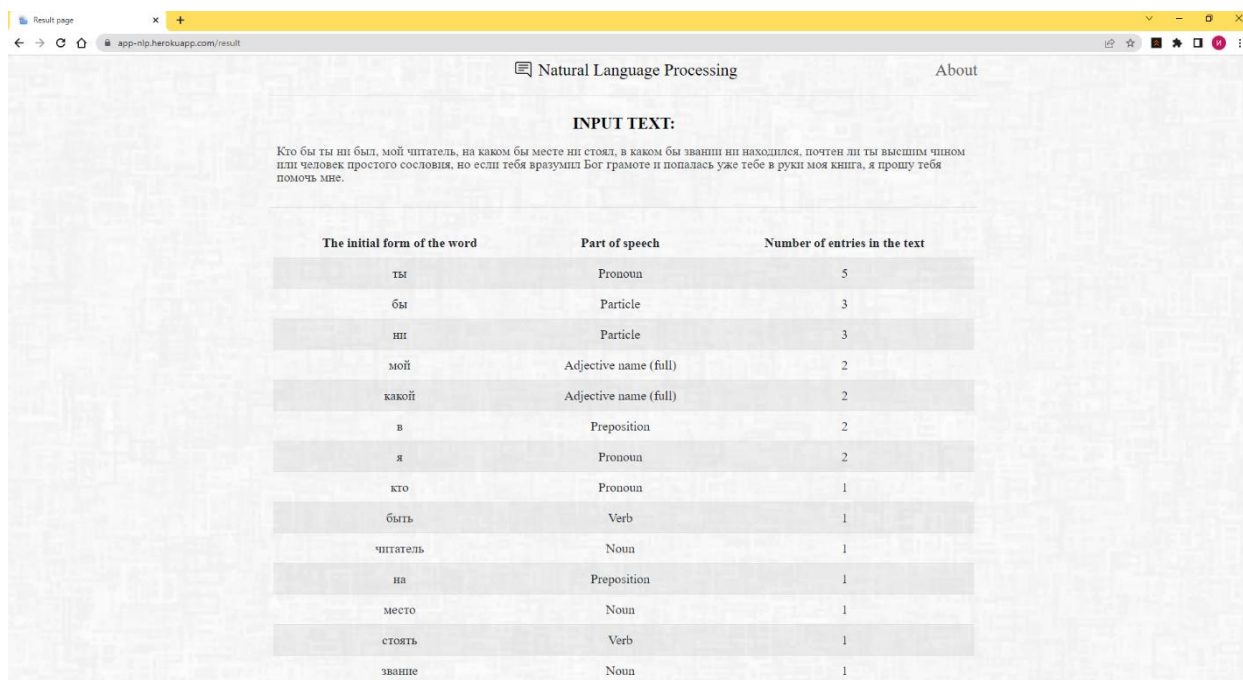


Рисунок 21 - Страница с информацией

Страница с результатом содержит исходный текст и таблицу с информацией о морфологическом анализе данного текста. В таблице три

столбца – Начальная форма слова, Часть речи, Количество вхождений в текст) (Рисунок – 21).



The initial form of the word	Part of speech	Number of entries in the text
ты	Pronoun	5
бы	Particle	3
ни	Particle	3
мой	Adjective name (full)	2
какой	Adjective name (full)	2
в	Preposition	2
я	Pronoun	2
кто	Pronoun	1
быть	Verb	1
читатель	Noun	1
на	Preposition	1
место	Noun	1
стоять	Verb	1
звание	Noun	1

Рисунок 22 - Страница с результатом

7 Задачи, выполненные в ходе разработки проекта.

В ходе разработке проекта были выполнены все поставленные задачи, а именно:

- Был написан алгоритм токенизации входного текста. Как по предложениям, так и по словам
- Был написан алгоритм обработки текста, который включает в себя: токенизацию исходного текста по словам, лемматизацию токенов и составления мешка слов, так же алгоритм определяет часть речи для каждой леммы
- Был разработан пользовательский интерфейс, который включает в себя: стартовую страницу, с возможностью загрузки текстового файла или написания текста в специальное поле, страницу с

информацией об авторе и назначении сайта, а также страницу, на которой отображается результат обработки текста в виде таблицы

- Приложение было загружено на хостинг

8 Тезариус

Фреймворк - программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Токенизация это процесс разбиения последовательности строк на части: на слова, ключевые слова, фразы, символы и другие элементы, которые называются токенами.

Стемминг - это процесс нахождения основы слова для заданного исходного слова. Основа слова не обязательно совпадает с морфологическим корнем слова.

Лемматизация — это процесс приведения словоформы к лемме — её нормальной (словарной) форме. В русском языке нормальными формами считаются следующие морфологические формы: для существительных — именительный падеж, единственное число; для прилагательных — именительный падеж, единственное число, мужской род; для глаголов, причастий, деепричастий — глагол в инфинитиве (неопределённой форме) несовершенного вида.

Web-интерфейс — это дизайн приложения, доступ к которому осуществляется через веб-браузер

9 Оценка степени завершенности и перспективы развития проекта.

Проект был полностью завершен и соответствует всем заданным условиям. В процессе разработки были решены все необходимые задачи, проблем при разработке не возникало. Данный проект имеет широкие перспективы для дальнейшего развития и может быть использован для машинного обучения.