СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Наименование программы

Название программы – «Русский учебный корпус» (англ «Russian learner corpus».)
Наименование программы для пользователя – «RLC».

1.2. Документы, на основании которых ведется разработка

Учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия".
Утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1. Функциональное назначение

Назначением приложения является загрузка текста с соответствующими метаданными в систему, аннотация загруженных текстов путем исправления ошибок и присвоения им тэгов, сбор статистики по загруженным текстам, а также выгрузка данных из приложения в формате JSON. Таким образом приложение является своего рода коллекцией большого количества текстов, написанных иностранцами на русском языке.

2.1.2. Эксплуатационное назначение

Данная программа была разработана для помощи иностранцам в изучении русского языка, а также сбора статистических данных, которые в дальнейшем могут применяться в анализе данных и машинном обучении. Сотрудники корпуса вручную размечают текста, исправляя ошибки и присваивая им советующий тег.

2.2. Краткая характеристика области применения

Данное ПО было разработано по запросу школы лингвистики НИУ ВШЭ в качестве замены существующей, но некорректно работающей программы. Смысл проекта заключается в сборе множества текстов, написанных иностранцами на русском языке, и дальнейшая их разметка. Как правило тексты загружаются практикантами. Помимо текста в корпус также загружается общая информация о тексте и его авторе. Сотрудники корпуса анализируют загруженный текст, находят и исправляют в нем ошибки, присваивая им соответствующие теги. В результате чего в базе данных проекта хранятся, как и оригинальный текст, так и все внесённые исправления. Среди представленных в корпусе текстов можно найти, как и обычные письменные тексты, так и так называемые “устные” тексты, представляющие собой запись устной речи в текстовом виде. В целом корпус может использоваться для помощи иностранцам в изучении русского языка. На основе большого количества данных, собираемых о текстах и их авторах можно проводить различные статистические исследования в области изучения русского языка. Также данные из этого проекта могут использоваться для создания датасетов, которые в дальнейшем могут применяться в области анализа данных и машинного обучения. В итоге наш продукт сможет заменить старую версию русского учебного корпуса, предлагая сотрудникам и посетителям более стабильную, удобную и простую в использовании версию сайта.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Изначально поставленной задачей было улучшение существующей версии русского учебного корпуса. Сотрудники корпуса жаловались на большое количество ошибок в работе программы, которые делали работу с корпусом практически невозможной. Так, например при исправлении ошибки в секции текста, программа могла “съесть” стоящие рядом слова. А некоторые страницы просто выбрасывали пользователю ошибку 500. Помимо этого, код, написанный изначальной командой разработчиков корпуса, был плохо задокументирован, обладал сильной связанностью, не использовал лучшие практики и был сложен для отладки. Стоит также отметить, что некоторые используемые библиотеки не обновлялись с 2016 года. Все вышеперечисленные факторы делали модификацию кода практически невозможной. Для исправления даже малейших ошибок приходилось переписывать целые блоки кода, разбросанные по разным файлам. После многочисленных попыток исправления ошибок в старой версии проекта, а также нескольких дистанционных встреч с сотрудниками корпуса, было приято решение полностью переписать программу с учетом выявленных нами недочетов и пожеланий пользователей. По запросу сотрудников корпуса, было решено добавить возможность сохранения авторов в базу данных. Это позволяет сотрудником не вносить каждый раз одну и ту же информацию, при добавлении нескольких текстов от одного и того же автора, а просто выбрать его имя из выпадающего списка. Однако позже, из-за большого количества текстов и авторов, представленных в корпусе, было принято решение сохранять не всех авторов, а только тех, кого захотят сохранить сами сотрудники. Это позволяет сохранить выпадающему списку относительно небольшой размер. Для реализации данного функционала нужно было спроектировать новую архитектуру базы данных, написать новые модели и сильно модифицировать старые. Также я решил выбрать PostgreSQL вместо MySQL в качестве используемой базе данных, т. к. он отличается большей производительностью и функциональностью. Одной из главных проблем в старом корпусе было использование огромного количества SQL запросов, которые очень сложны в отладке, особенно в таком количестве. Поэтому было принято решение перейти на ORM запросы, которые куда более просты в отладке и общем понимании. Другой проблемой в старом корпусе было отображение вкладки “Статистика”, которая содержала большое количество информации в достаточно сложном для понимания виде. Поэтому было принято решение сделать статистику более наглядной, путем визуализации данных в виде графиков. Мною были подготовлены все необходимые данные для построения графиков и диаграмм, которые в дальнейшим были построены совместно с другим членом команды. Помимо графиков, мною также было добавлена таблица с основной информацией о данных корпуса. Чтобы улучшить пользовательский опыт, было принято решение сделать несколько простых анимаций, делающий сайт более “живым”.

Поставленные задачи должны выполнены:

1) Написание моделей для отображения и хранения данных
2) Проектирование архитектуры базы данных проекта для хранения информации пользователей, документов, авторов и главной страницы
3) Токенизация и морфологический разбор текстов с применением машинного обучения
4) Написание ORM запросов к базе данных для получения и изменения данных
5) Сбор, обработка и отображение статистических данных проекта
6) Создание анимаций для статических элементов страницы

3.2. Описание алгоритма и функционирования программы

3.2.1. Отображение и хранение в базе данных

Помимо встроенных моделей Django для хранения объектов в базе данных также мною были разработаны следующие модели:

1) Author – модель автора, хранящая в себе поля с информацией об авторе каждого текста (Например: доминантный язык, пол автора, имя автора и программа, на которой он обучается, уровень знания языка и т. д.)
2) Document – модель для хранения информации о тексте (сам текст и метаданные о нем)
3) Sentence – модель для хранения предложений в тексте (ссылка на текст (foreign key), само предложение, HTML разметка с морфологическим разбором и номер предложения в тексте)
4) Section – модель для секции главной страницы

3.2.2. Архитектура базы данных для хранения информации пользователей, документов, авторов и главной страницы

Структура базы данных (Рисунок 1):

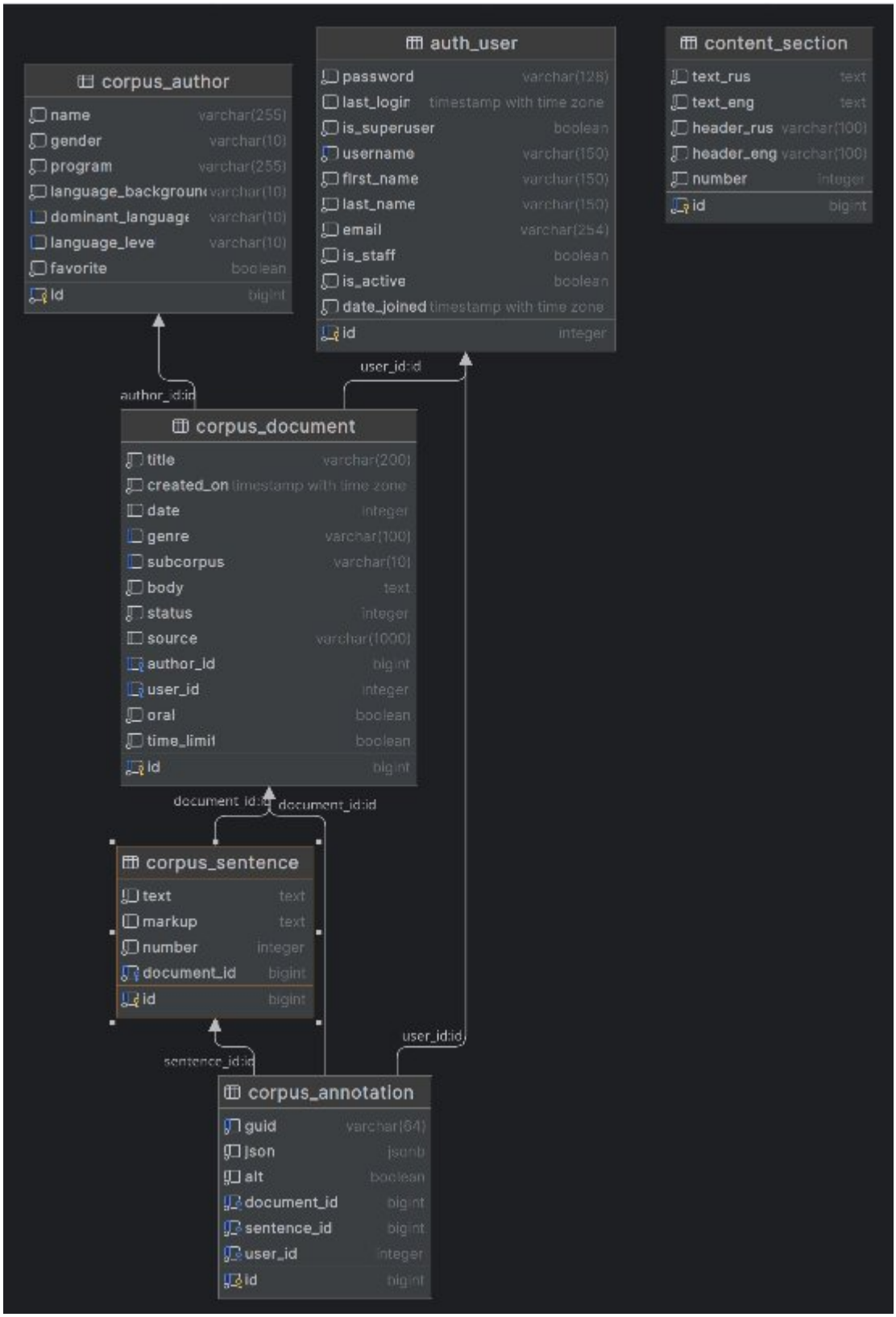


Рисунок 1. Архитектура базы данных

1) auth_user – информация о зарегистрированном пользователе
2) corpus _author – информация об авторе текста
3) content_section – информация на главной странице

4) corpus_document – информация о документах
5) corpus_sentence – информация о предложениях

Т. к. старая версия корпуса не поддерживала функционал сохранения автора, архитектуру базы данных пришлось сильно изменить. Сохранение автора позволяет сэкономить время при добавлении нескольких текстов, написанных одним и тем же человеком, т. к. сотруднику не придется заполнять одни и те же поля несколько раз. Достаточно будет просто выбрать нужного автора из выпадающего меню. Поскольку случаи, когда в корпус сразу заносятся несколько текстов от одного и того же человека встречаются не так часто, в корпусе представлено большое количество текстов только с одним автором. Добавление таких авторов в выпадающее меню является бессмысленным и только ухудшает его читабельность. Поэтому сотрудники корпуса сами решают, каких авторов сохранять в меню, а каких нет. Но тогда возникает вопрос, как оптимально и без дублирования информации хранить данные, если не у всех документов есть сохранённый автор. Важно учитывать, что добавить документ без информации об его авторе нельзя. Как тогда хранить эти данные? Для решения данной проблемы было придумано использовать автора, как своего рода “контейнер” с информацией. Так, когда сотрудник добавит новый текст, но решит не сохранять автора для дальнейшего использования, в таблице с авторами все равно появится новая запись. Создастся так называемый “анонимный” автор. Он будет служить для хранения информации об авторе текста, но не будет высвечиваться в выпадающем меню авторов.

3.2.3. Морфологический разбор и токенизация

Для работы с токенизацией и морфологическим разбором была встроена библиотека Natasha, которая объединяет несколько инструментов для анализа текстов и использует технологии машинного обучения. Пример кода:

```
# load models
segmenter = Segmenter()
morph_vocab = MorphVocab()
emb = NewsEmbedding()
morph_tagger = NewsMorphTagger(emb)
doc = Doc(self.body)

# tokenize the text
doc.segment(segmenter)

# tag morphology
doc.tag_morph(morph_tagger)

# lemmatize all tokens
for token in doc.tokens:
    token.lemmatize(morph_vocab)
```

При добавлении/сохранении текста он проходит предобработку, где удаляются все повторяющиеся пробелы. Это нужно для корректной работы инструментов аннотатора. Если документ был сохранен после редактирования, то из базы данных удаляются все связанные с ним аннотации. Далее при помощи Natasha текст разбивается на токены (слова), производится морфологический разбор и лемматизация каждого токена. На основе полученной информации генерируется HTML разметка, для отображения на странице.

3.2.4. Запросы для получения и изменения данных

Первоначальный проект включал в себя множество SQL запросов, которые сложно поддерживать и тестировать. Поэтому было принято решение переписать их в более удобном виде

с помощью ORM запросов. Помимо перечисленных отличий они являются более читаемыми и безопасными. Примеры изменений:

Было (SQL запрос):

```
def exact_search(word, docs, flag, expand, page, per_page):
    db = Database()
    word = word.split()[0]
    req1 = (
        'SELECT COUNT(DISTINCT doc_id) FROM `annotator_token` WHERE token="'
        + word
        + '" '
    )
    if flag:
        req1 += "AND doc_id IN (" + ",".join(docs) + ");"
    docs_len = int(db.execute(req1)[0][0])
    n_req = (
        'SELECT COUNT(DISTINCT sent_id) FROM `annotator_token` WHERE token="'
        + word
        + '" '
    )
    if flag:
        n_req += "AND doc_id IN (" + ",".join(docs) + ");"
    sent_num = int(db.execute(n_req)[0][0])
    req2 = 'SELECT DISTINCT sent_id FROM `annotator_token` WHERE token="' + word
+ '" '
    if flag:
        req2 += "AND doc_id IN (" + ",".join(docs) + ")"
    req2 += " LIMIT %d,%d;" % ((page - 1) * per_page, per_page)
    sentences = "(" + ", ".join([str(i[0]) for i in db.execute(req2)]) + ")"
    if sentences != "()":
        req3 = (
            'SELECT sent_id, num FROM `annotator_token` WHERE token="'
            + word
            + '" AND sent_id IN '
            + sentences
        )
        tokens = db.execute(req3)
    else:
        tokens = []
    # tokens = Token.objects.filter(token__exact=word)
    e = defaultdict(list)
    for i, j in tokens:
        e[i].append(j)
    jq = []
    sent_list = [ShowSentence(i, e[i], expand) for i in sorted(e)]
    ShowSentence.empty()
    for sent in sent_list:
        jq.append(jquery.replace("***", str(sent.id)))
    return jq, sent_list, word, docs_len, sent_num
```

Стало (ORM запрос):

```
def body_search(self, queryset, name, value):
        return queryset.filter(body__search=value)
...
    body = Django_filters.CharFilter(
        method="body_search", label=_("Body search")
    )
```

3.2.5. Статистические данные

Схема передачи данных для работы статистики (Рисунок 2):

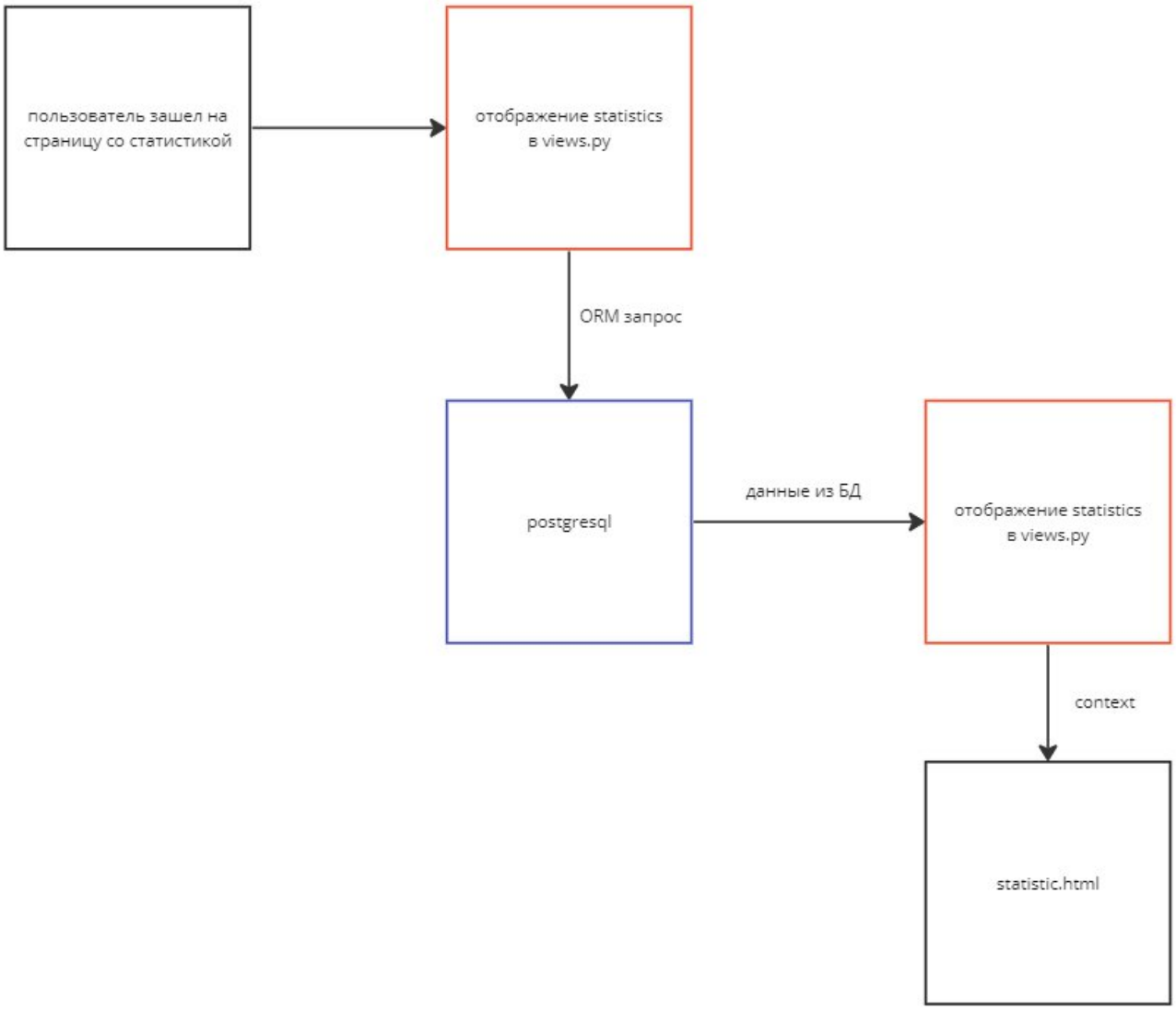
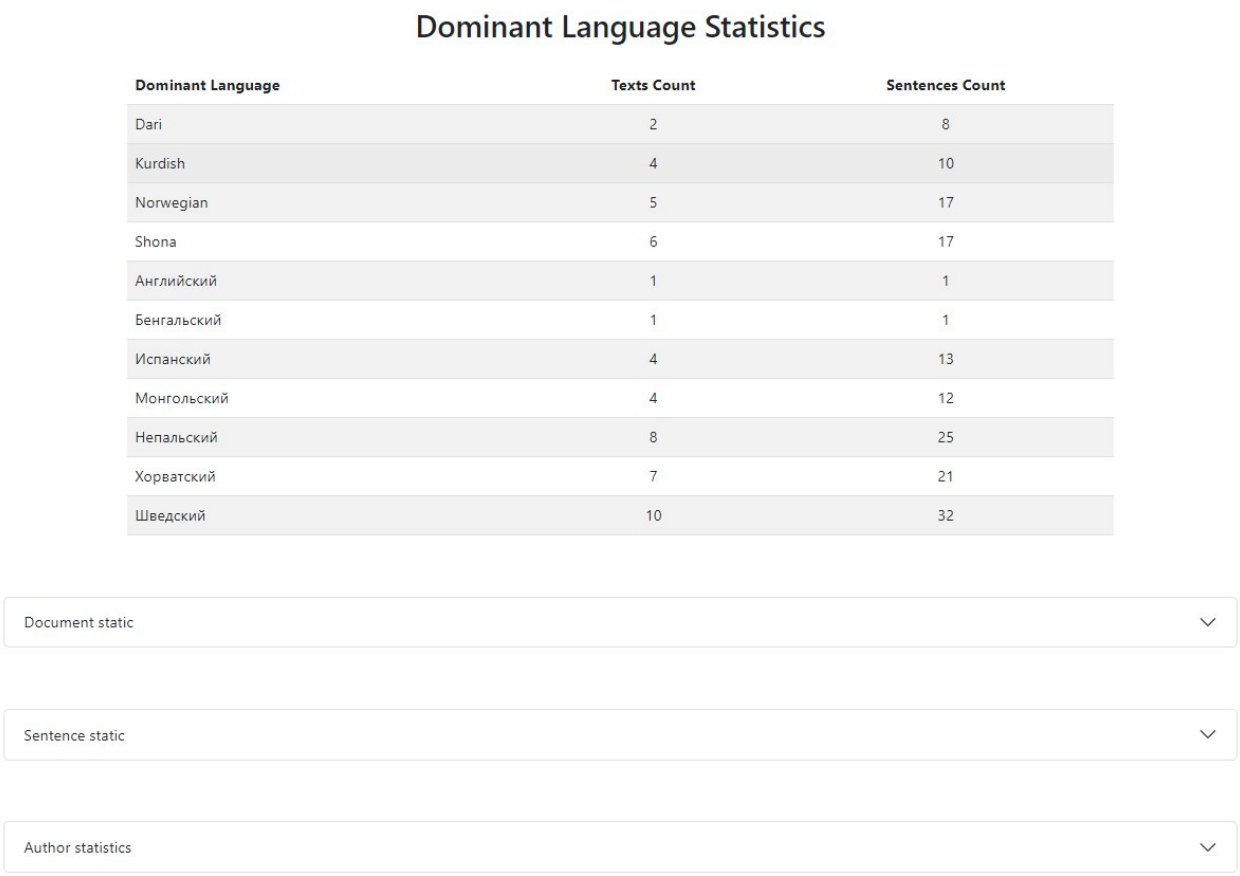


Рисунок 2. Схема передачи данных для статистики

1) Пользователь заходит на страницу “Статистика”, тем самым вызывая ее отображение (функцию statistics, объявленную в views.py)
2) statistics формирует и отправляет ORM запрос в базу данных, тем самым получая данные для построения статистики
3) Данные через context передаются в statistics.html, где при помощи javascript (библиотеки char.js) строятся графики

В итоге при заходе на страницу пользователь видит следующее (Рисунок 3). В Каждом аккордеоне представлена более детальная статистика в виде ряда графиков.



Dominant Language Statistics

Dominant Language	Texts Count	Sentences Count
Dari	2	8
Kurdish	4	10
Norwegian	5	17
Shona	6	17
Английский	1	1
Бенгальский	1	1
Испанский	4	13
Монгольский	4	12
Непальский	8	25
Хорватский	7	21
Шведский	10	32

Рисунок 3. Страница со статистикой

3.2.6. Анимирование результатов статистики

Для эстетического эффекта были добавлены анимации подсчета итогового количества документов, авторов и предложений. Также была добавлена анимация повеления таблицы при загрузи страницы. Вот пример JavaScript функции, используемой для анимации счетчика текстов:

```
function animateCounter(counterId, targetAccordion) {
    let totalValue = parseInt(document.querySelector(counterId).textContent);
    const accordionButton = document.querySelector(targetAccordion);
    accordionButton.addEventListener('show.bs.collapse', () => {
        totalValue = parseInt(document.querySelector(counterId).textContent);
    });
   let speed = 5;
   if (totalValue > 100) {
      let counter = totalValue - 100;
    } else if (totalValue > 50) {
      let counter = 0;
      speed = 15;
    } else {
      let counter = 0;
      speed = 40;
    }

    const counterElement = document.querySelector(counterId);
    let intervalId = null;
    let isAccordionOpen = false;
```

```
    accordionButton.addEventListener('click', () => {
        if (accordionButton.getAttribute('aria-expanded') === 'true') {
            isAccordionOpen = true;
            clearInterval(intervalId);
            if (totalValue > 100) {
                counter = totalValue - 100;
            } else {
                counter = 0;
            }
            intervalId = setInterval(() => {
                counterElement.textContent = counter;
                counter++;
                if (counter > totalValue) {
                    clearInterval(intervalId);
                    counterElement.textContent = totalValue;
                }
            }, speed);
        } else {
            isAccordionOpen = false;
        }
    });
}
```

Данная функция проигрывает анимацию подсчета от числа, на 100 меньше итогового, до непосредственного того числа, которое нужно отобразить. Также предусмотрен случай, когда итоговое число меньше 100. В таком случае подсчет будет идти от 0 до итогового числа с меньшей скоростью. Данная анимация проигрывается каждый раз при открытии аккордеона.

3.3. Описание и обоснование выбора метода организации входных и выходных данных

3.3.1. Описание метода организации входных и выходных данных

Входные данные представляют собой запросы REST API, а также документы, добавленные в корпус. Выходные данные: Отображение соответствующей страницы сайта, статистическое отображение информации о собранных данных в виде таблицы и графиков, а также выгрузка текстов и их аннотаций в формате JSON.

3.3.2. Обоснования выбора метода организации входных и выходных данных

JSON — это легкий и распространенный формат обмена данными между приложениями. Существуют многочисленные причины, почему мною был выбран JSON. Во-первых, JSON легко читается и понимается как человеком, так и компьютером. Он использует простой и интуитивно понятный синтаксис, основанный на объектах и массивах, что делает его гораздо более читаемым, чем другие форматы данных, такие как XML. Во-вторых, JSON поддерживается большинством языков программирования и имеет встроенные библиотеки для работы с данными в различных языках, что делает его легко доступным для разработчиков и позволяет быстро и без проблем интегрировать его в любой проект. Это облегчит дальнейшую работу с данными из корпуса после их выгрузки. Наконец, JSON также обладает отличной масштабируемостью и позволяет легко добавлять новые поля и свойства в объекты без необходимости изменения всей структуры данных. Это может быть полезно в ситуации, если в корпусе решат собирать новый вид данных.

3.4. Описание и обоснование выбора состава технических и программных средств

3.4.1. Состав технических и программных средств

Процессор: 1 ГГц или более быстрый процессор.

Оперативная память (RAM): не менее 2 ГБ.
Жесткий диск: свободное место на диске не менее 500 МБ.
Видеокарта: поддержка графики с разрешением не менее 1024x768 пикселей.
Операционная система: Windows 7 или более поздняя версия, MacOS, Linux или другие поддерживаемые операционные системы.
Браузер: последняя версия любого из популярных браузеров, таких как Google Chrome, Mozilla Firefox, Apple Safari или Microsoft Edge.

3.4.2. Обоснование выбора технических и программных средств

Описанные характеристики представляют минимальные требования для работы с веб-браузером и основываются на общих рекомендациях и практике.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данной работы расчет экономической эффективности не предусмотрен.

4.2. Предполагаемая потребность

Любой пользователь, у которого есть веб-браузер, выход интернет и устройство, удовлетворяющее минимальным техническим характеристикам, может использовать данное приложение.

Потребность в данном продукте имеют люди, изучающий русский язык как иностранный, эритажные носители, а также исследователи в области освоения иностранных языков и преподаватели.

Данное ПО будет особенно полезно людям, заинтересованным в изучении русского языка, а также исследователям, статистам и тем, кто занимается машинным обучением в области распознавания и классификации ошибок в тексте, написанном на русском языке.

4.3. Экономические преимущества разработки по сравнению с отечественными и зарубежными образцами или аналогами

Корпус – это не просто набор неправильно написанных текстов, это огромная и уникальная коллекция данных, содержащих в себе тысячи примеров тех или иных ошибок, а также путей их исправления. Каждая ошибка имеет собственную классификацию, что позволяет легко упорядочивать и анализировать представленную информацию. Пользователь может на конкретных примерах узнать о различных видах ошибок в русском языке. А благодаря большому количеству данных, пользователь способен увидеть так называемые “подводные камни” русского языка, увидев в каких аспектах русского чаще всего допускаются ошибки. Дополнительная информация, собираемая о текстах и их авторах позволяет проводить различные статистические исследования. Например, какие части русского языка сложнее даются тем или иным языковым группам. Русский учебный корпус – это по-настоящему уникальный проект, не имеющий аналогов на отечественном рынке. Однако если говорить про сферу обучения русскому языку, то можно привести ряд других платформ, обладающих той же целью: Dualingo, Babbel, russianforfree.

5. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1) ГОСТ 19.101-77 Виды программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
2) ГОСТ 19.102-77 Стадии разработки. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
3) ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
4) ГОСТ 19.104-78 Основные надписи. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
5) ГОСТ 19.105-78 Общие требования к программным документам. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
8) ГОСТ 19.603-78 Общие правила внесения изменений. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.
9) ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. //Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

10) Django documentation [Электронный ресурс]. Режим доступа: https://docs.djangoproject.com/en/4.2/, свободный (Дата обращения 10.05.2023)
11) Bootstrap documentation [Электронный ресурс]. Режим доступа: https://getbootstrap.com/docs/5.3/getting-started/introduction/, свободный (Дата обращения 10.05.2023)
12) jQuery API Reference [Электронный ресурс]. Режим доступа: https://api.jquery.com/, свободный (Дата обращения 10.05.2023)
13) RecogitoJS API Reference [Электронный ресурс]. Режим доступа: https://github.com/recogito/recogito-js/wiki/API-Reference, свободный (Дата обращения 10.05.2023)
14) Docker documentation [Электронный ресурс]. Режим доступа: https://docs.docker.com/get-started/, свободный (Дата обращения 10.05.2023)
15) Web annotation data model [Электронный ресурс]. Режим доступа: https://www.w3.org/TR/annotation-model/, свободный (Дата обращения 10.05.2023)
16) RLC [Электронный ресурс]. Режим доступа: http://web-corpora.net/RLC, свободный (Дата обращения 10.05.2023)

17) Natasha — качественное компактное решение для извлечения именованных сущностей из новостных статей на русском языке [Электронный ресурс]. Режим доступа: https://natasha.github.io/ner/, свободный (Дата обращения 10.05.2023)

ПРИЛОЖЕНИЕ 1

ТЕРМИНОЛОГИЯ

Ниже приведен список необходимых терминов для ознакомления.

Django - это бесплатный и открытый фреймворк для создания веб-приложений на языке Python.

Метаданные – это данные, которые описывают характеристики других данных, такие как формат, дата создания или автор.

Аннотация – исправление ошибки в тексте с добавлением соответствующего тега и комментария

JSON – это легкий формат обмена данными, основанный на синтаксисе объектов JavaScript, который часто используется для передачи данных между клиентом и сервером в сети Интернет.

SQL запрос - команда, которая направляется к базе данных, чтобы получить или изменить данные в соответствии с определенными условиями

ORM запрос – запрос, использующийся для извлечения, изменения или удаления данных из реляционной базы данных

Датасет – это набор данных, обычно представленный в табличной форме, который используется для анализа и машинного обучения.

REST API – это интерфейс программирования приложений, который использует протокол HTTP для обмена данными и позволяет взаимодействовать с удаленными серверами по принципу "ресурс-действие".

Natasha – коллекция библиотек на языке python, предназначенная для анализа и обработки текстов на русском языке.

PostgreSQL - это бесплатная и мощная система управления реляционными базами данных (СУБД), которая поддерживает расширяемость, SQL-стандарты и множество функциональных возможностей.

MySQL - это бесплатная и популярная система управления реляционными базами данных (СУБД), используемая для хранения и управления данными веб-приложений и других программных продуктов.

Chart.js – это библиотека JavaScript, которая позволяет создавать анимированные и интерактивные графики, диаграммы и графы на веб-страницах.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц в докум.)	№ документа	Входящий № сопроводительного докум. и дата	Подп.	Дата
Изм.	Измененных	Замененных	Новых	Аннулированных					