СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Наименование программы

Наименование программы – «Русский учебный корпус» или «Russian learner corpus».
Наименование программы для пользователя – «RLC».

1.2. Документы, на основании которых ведется разработка

Основанием для разработки является учебный план подготовки бакалавров по направлению 09.03.04 "Программная инженерия" и утвержденная академическим руководителем тема курсового проекта.

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение программы

2.1.1. Функциональное назначение

Функциональным назначением приложения является добавление текстов в систему и их дальнейшая разметка с выявлением и исправлением ошибок, а также просмотр уже размеченных текстов. Также данные с сайта могут использоваться в исследовании области изучения иностранных языков.

2.1.2. Эксплуатационное назначение

Эксплуатационным назначением программы является помощь пользователю, изучающему русский язык как иностранный, или являющимся эритажным носителем языка. Для этого на сайт загружено множество текстов для подробного изучения правил русского языка на основе ошибок в этих текстах.

2.2. Краткая характеристика области применения

Программа RLC была разработана по запросу Школы лингвистики ВШЭ для замены существующей системы разметки текстов. В Русском учебном корпусе содержатся образцы устной и письменной речи двух категорий нестандартных говорящих на русском языке: изучающих русский язык как иностранный и эритажных говорящих (унаследовавших знание языка в детстве). Корпус позволяет производить поиск по лексико-грамматическим свойствам, а также по самым разным типам отклонений от стандартной русской речи - от орфографических ошибок до выбора лексических единиц и конструкций.

Представленные тексты анализируются работниками корпуса на предмет ошибок и неточностей, для чего им предоставляется соответствующий интерфейс. Обычные пользователи имеют доступ к содержимому текста и аннотаций к нему.

Данные, находящиеся в корпусе, представляют особую ценность для задач машинного обучения, поскольку являются уникальными и могут использоваться для анализа текстов на предмет ошибок.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Старая версия системы имела множество недочетов и критических ошибок, делающих работу с сайтом затруднительной. На основании опыта анализа предыдущей версии было принято решение о написании полностью новой системы, справляющейся с поставленной задачей лучше.

Одни из основных проблем старой версии Русского учебного корпуса, над которыми была предпринята работа, перечислены ниже:

1) Некорректная работа системы аннотаций – в системе аннотаций присутствует множество ошибок, которые приводят к некорректному результату (например, аннотации индексируются по словам, а не по символам, из-за чего аннотации могут «съехать» влево или вправо)
2) Устаревшая система развертки проекта на сервере
3) Неэффективная работа бэкенд-части приложения
4) Запутанная и немодульная архитектура приложения

Поставленные задачи должны быть выполнены:

1) Реализовать авторизацию и регистрацию пользователей на сайте
2) Контейнеризация приложения (развертывание с помощью отдельных Docker контейнеров и их разделение на задачи)
3) Внедрение, модификация и настройка библиотеки для аннотации
4) Написание функционала для добавления, редактирования, аннотирования и удаления текстов
5) Добавление инструмента автоматических аннотаций с использованием машинного обучения
6) Выгрузка аннотаций и текстов из базы данных в формате JSON
7) Фильтрация текстов и контекстный поиск
8) Разработка API сайта для взаимодействия бэкэнда и фронтэнда

3.2. Описание алгоритма и функционирования программы

3.2.1. Регистрация и авторизация пользователя

Поскольку основная бэкэнд-часть проекта написана на Django, а встроенная в него авторизация и регистрация пользователя полностью удовлетворяет всем запросам разработки, было принято решение использовать именно ее.

3.2.2. Схема контейнеризации

На схеме представлено 3 контейнера:

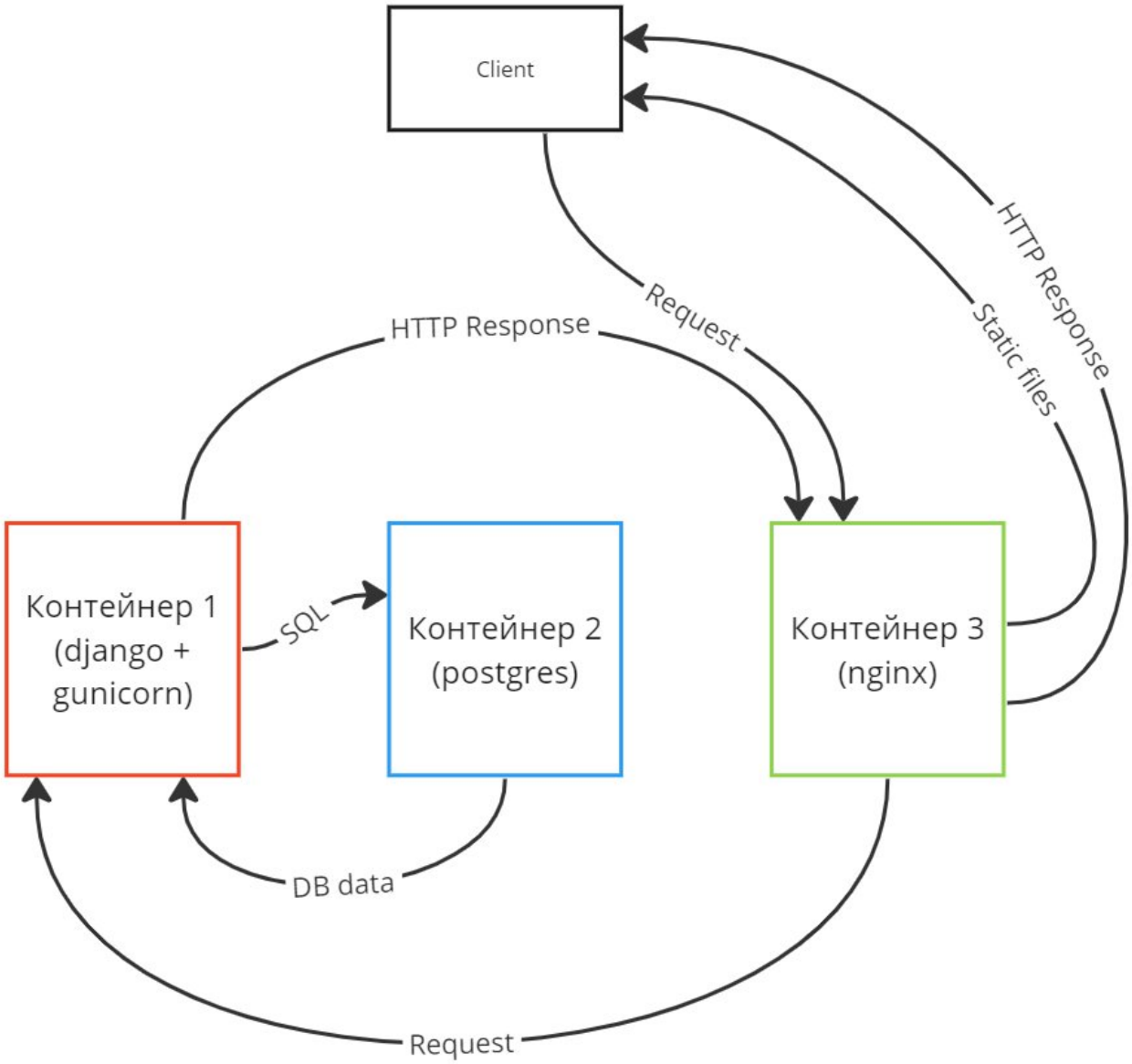

Рис. 1 -

Django + gunicorn: основной контейнер приложения, который запускает код Django. Django приложение может обращаться к базе данных Postgres и слушать входящие HTTP запросы.

Postgres: контейнер базы данных. Django контейнер обращается к этому контейнеру для выполнения любых операций с базой данных, таких как создание, чтение, обновление и удаление данных (CRUD операции).

Nginx: контейнер веб-сервера. Он принимает входящие HTTP запросы и перенаправляет их на контейнер Django. Nginx используется для обслуживания статических файлов, таких как CSS и JavaScript.

В файле docker-compose.yml описано взаимодействие контейнеров:

1) Входящий HTTP запрос приходит на сервер Nginx.
2) Nginx перенаправляет запрос на контейнер Django.

3) Django обрабатывает запрос. Если для обработки запроса требуется обращение к базе данных, Django отправляет запрос в контейнер Postgres.
4) Postgres выполняет операции с базой данных и возвращает результат Django.
5) Django формирует HTTP ответ и отправляет его обратно на Nginx.
6) Nginx отправляет HTTP ответ клиенту.

3.2.3. Работа с библиотекой для аннотации

Библиотека RecogitoJS (https://github.com/recogito/recogito-js) была модифицирована и настроена для работы с коррекцией ошибок. Список изменений:

1) Добавление тегов для ошибок посредством настройки виджета `comment`:

```
{
  widget: 'TAG',
  vocabulary: ['Graph', 'Hyphen', 'Space', 'Ortho', 'Translit', 'Misspell',
'Deriv', 'Infl', 'Num', 'Gender', 'Morph', 'Asp', 'ArgStr', 'Passive', 'Refl',
'AgrNum', 'AgrCase', 'AgrGender', 'AgrPers', 'AgrGerund', 'Gov', 'Ref', 'Conj',
'WO', 'Neg', 'Aux', 'Brev', 'Syntax', 'Constr', 'Lex', 'CS', 'Par', 'Idiom',
'Transfer', 'Not-clear', 'Del', 'Insert', 'Transp', 'Subst', 'Altern', 'Tense',
'Mode']
},
```

2) Добавление виджета для дополнительных аннотаций

```
var CheckboxWidget = function (args) {

  // 1. Find a current check status in the annotation, if any
  var currentCheckBody = args.annotation ?
    args.annotation.bodies.find(function (b) {
      return b.purpose == 'highlighting';
    }) : null;

  // 2. Keep the value in a variable
  var currentCheckValue = currentCheckBody ? currentCheckBody.value : false;

  // 3. Triggers callbacks on user action
  var toggleCheck = function (evt) {
    if (currentCheckBody) {
      args.onUpdateBody(currentCheckBody, {
        type: 'TextualBody',
        purpose: 'highlighting',
        value: evt.target.checked
      });
    } else {
      args.onAppendBody({
        type: 'TextualBody',
        purpose: 'highlighting',
        value: evt.target.checked
      });
    }
  }

  // 4. This part renders the UI elements
  var createCheckbox = function (checked) {
    var checkbox = document.createElement('input');
```

```
    checkbox.type = 'checkbox';
    checkbox.checked = checked;
    checkbox.addEventListener('change', toggleCheck);
    return checkbox;
  }
  var createLabel = function () {
    var label = document.createElement('label');
    label.textContent = 'Дополнительная аннотация';

    return label;
  }

  var container = document.createElement('div');
  container.className = 'checkbox-widget';

  var checkbox = createCheckbox(currentCheckValue);
  var label = createLabel();

  container.appendChild(checkbox);
  container.appendChild(label);

  return container;
}
```

3) Настройка для работы с API сайта (получение/создание/изменение/удаление аннотаций)

```
r.setAuthInfo(data);

if (sentence.dataset.alt === 'true') {
  r.loadAnnotations(`/api/annotations/get/alt/${sentence.dataset.sentenceId}/`);
} else {
  r.loadAnnotations(`/api/annotations/get/${sentence.dataset.sentenceId}/`);
}

r.on('createAnnotation', async (annotation, overrideId) => {
  // TODO проверять, что аннотации не залезают друг на друга
  // Посылаем POST-запрос на сервер для сохранения аннотации
  await createAnnotation(
    sentence.dataset.documentId,
    sentence.dataset.sentenceId,
    sentence.dataset.userId,
    annotation.id,
    sentence.dataset.alt,
    annotation
  );
  console.log('Stored annotation:', annotation);
  refreshCorrections();
});
r.on('updateAnnotation', async (annotation, previous) => {
  // Посылаем POST-запрос на сервер для сохранения аннотации
  await updateAnnotation(
    sentence.dataset.documentId,
    sentence.dataset.sentenceId,
    sentence.dataset.userId,
    annotation.id,
```

```
    sentence.dataset.alt,
    annotation
  );
  console.log('Stored annotation:', annotation);
  refreshCorrections();
});
r.on('deleteAnnotation', async (annotation) => {
  // Посылаем POST-запрос на сервер для удаления аннотации
  await deleteAnnotation(annotation.id);
  console.log('Deleted annotation:', annotation);
  refreshCorrections();
});
```

3.2.4. Инструменты для аннотирования текстов

После добавления текста в базу данных пользователю становится доступен интерфейс аннотирования. Для создания аннотации необходимо выделить фрагмент текста, содержащий ошибку, и заполнить необходимые поля (исправление, комментарий (опционально), тэги ошибок, дополнительная/основная аннотация (дополнительная аннотация не отображается в скорректированной версии предложения)).
Интерфейс для аннотирования текста выглядит следующим образом:

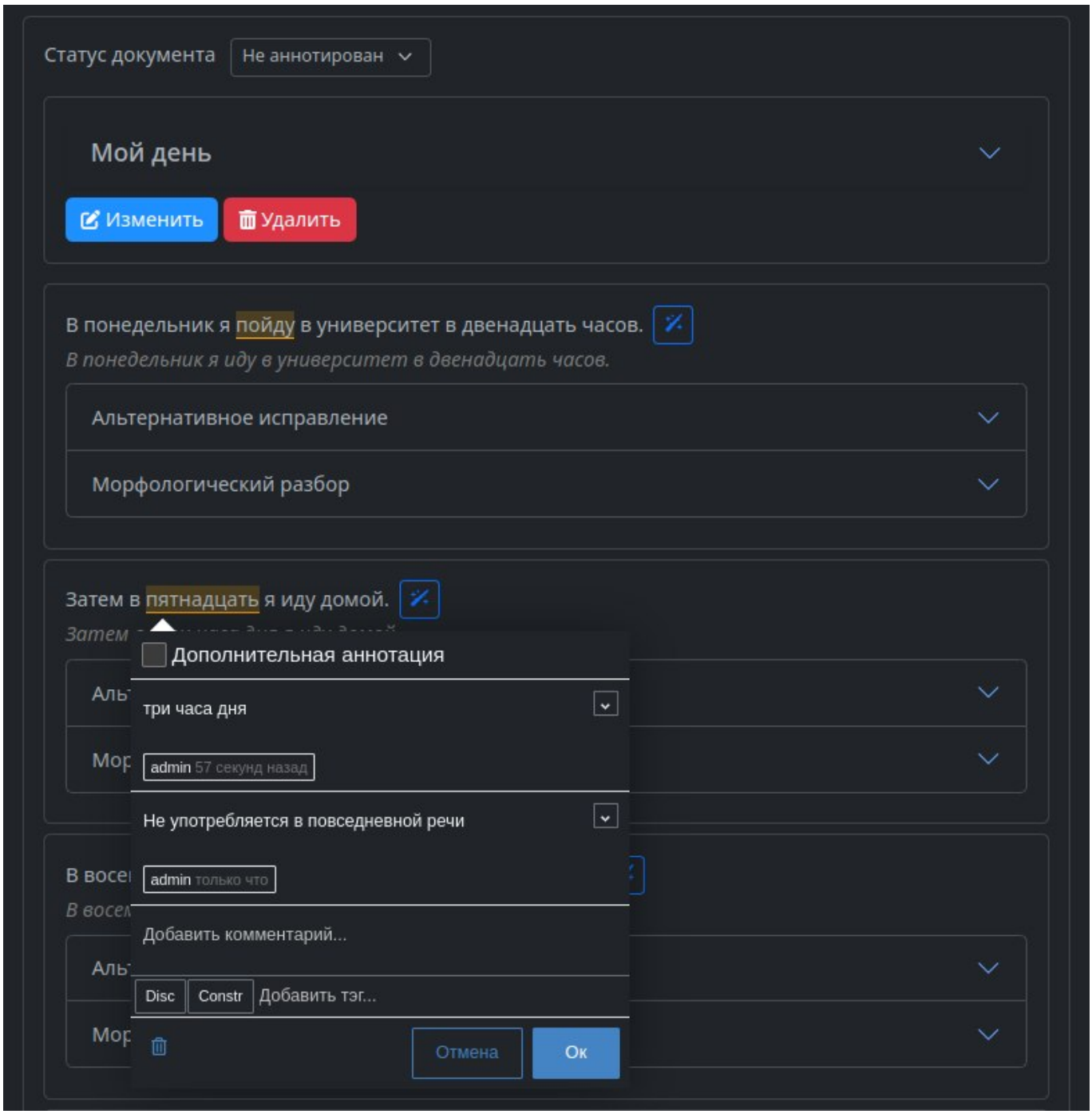

Рисунок 2

После нажатия кнопки “Ок” аннотация сохраняется и заносится в базу.
Дополнительные возможности аннотирования:

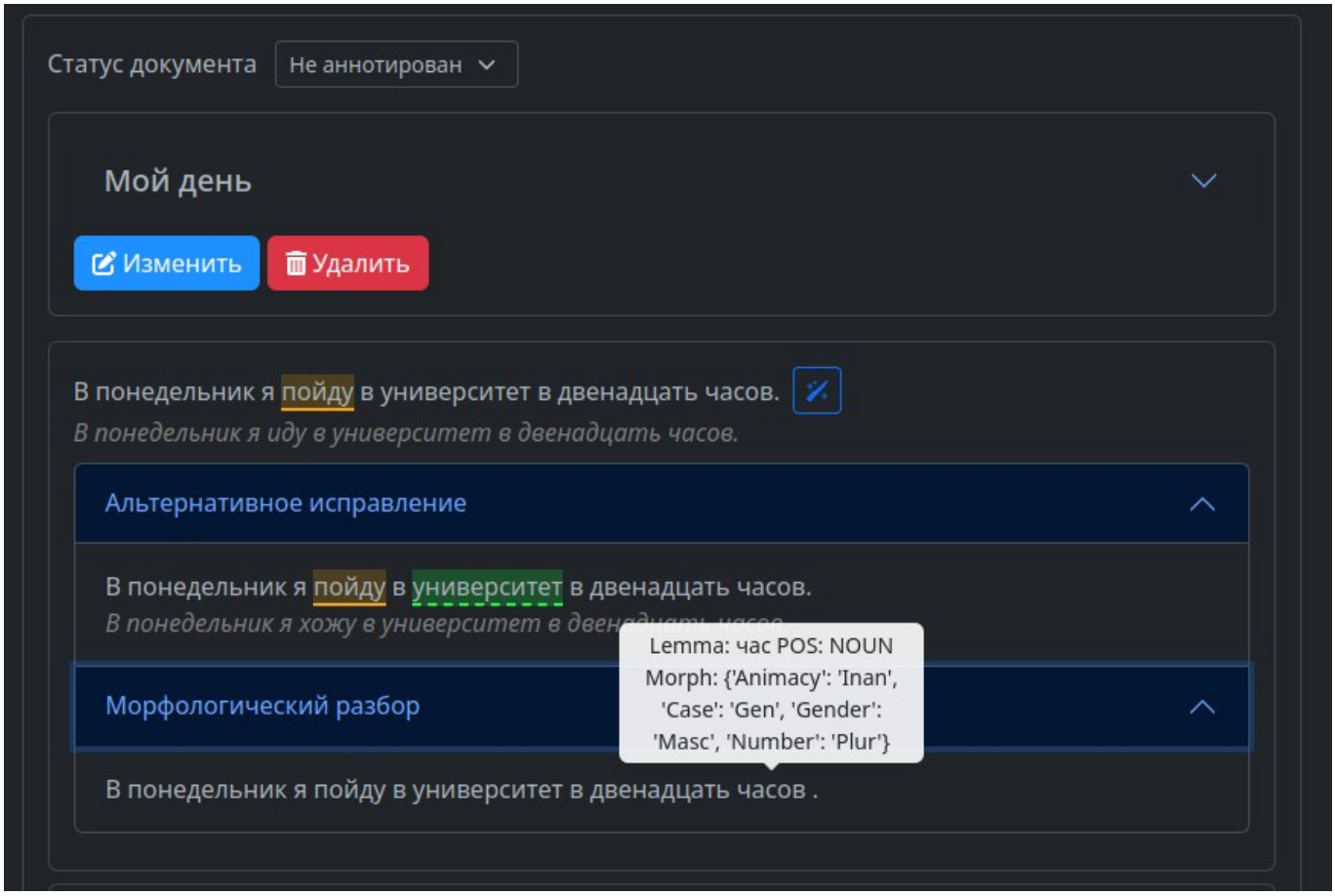

Рисунок 3

Помимо основного функционала пользователь получает такие инструменты, как морфологический разбор и альтернативная аннотация. Аналогично, по кнопке “Ок” изменения переносятся в базу данных.

Альтернативная аннотация – второй вариант исправления предложения (например, слово пойду можно заменить на иду или хожу).

Морфологический разбор – при наведении курсора мыши на слово показывается всплывающее окно с информацией о морфологических характеристиках слова.

Также благодаря автоматической аннотации доступна возможность изначально написать исправленное предложение, а после система сама проставит аннотацию и теги ошибок:

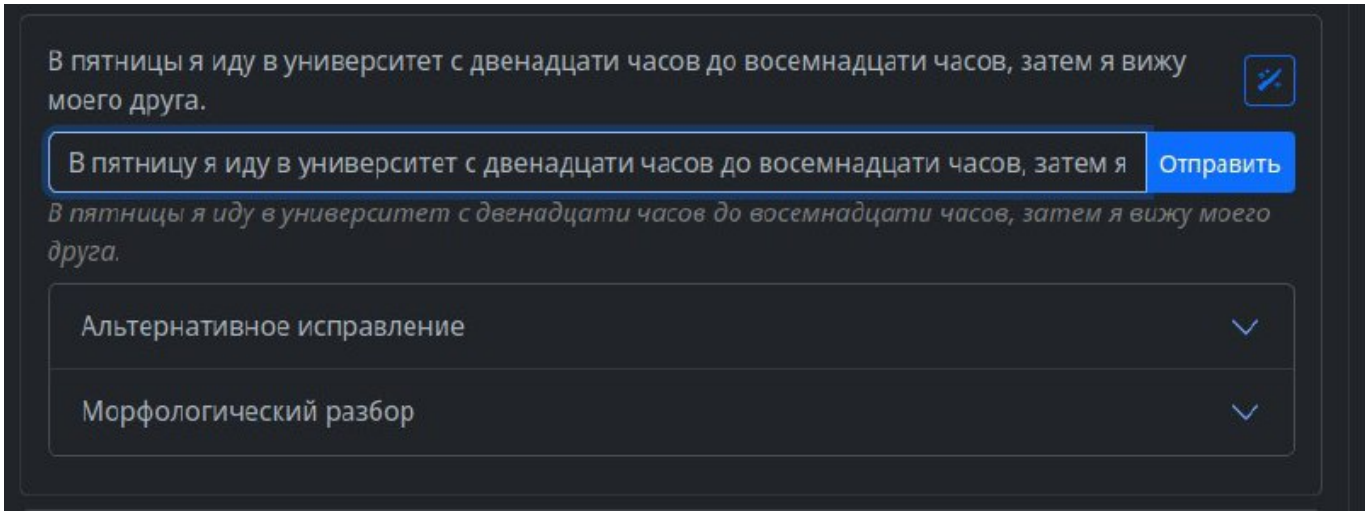

Рисунок 4

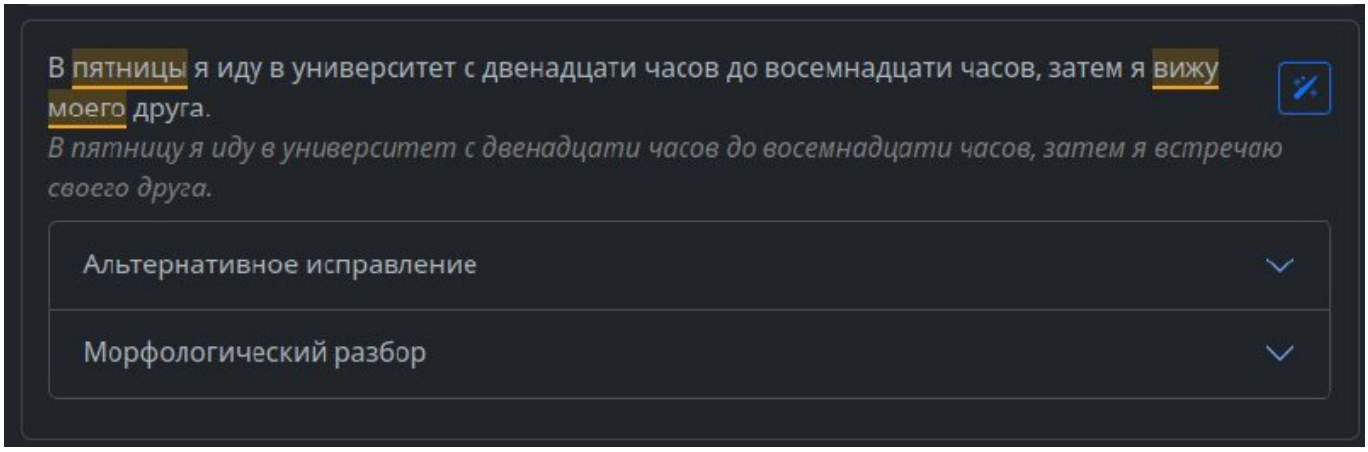

Рисунок 5

3.2.5. Машинное обучение в аннотации текстов

Для автоматической разметки текстов используется библиотека Annotator (https://github.com/objectivcp/annotator). При вводе исправленного предложения она автоматически создает аннотации с правильными словоформами и расставленными тегами ошибок:

```
def auto_annotate(request):
    if request.method == "POST":
        original = request.POST["original_sentence"]
        corrected = request.POST["corrected_sentence"]
    else:
        return JsonResponse({"error": "Only POST requests are allowed."})

    a = Annotator()
    edits, orig_tokenized, cor_tokenized = a.annotate(original, corrected)
    annotations = []
    for index, edit in enumerate(edits):
        original_tokens = edit.o_toks
        if len(original_tokens) == 0:
            text, tokens, number = (
                original,
                [tok.text for tok in orig_tokenized.tokens],
                edit.o_start,
```

```
            )
            original_start, original_end = get_word_positions(text, tokens,
number)
        else:
            original_start = original_tokens[0].start
            original_end = original_tokens[-1].stop

        guid = uuid.uuid4()
        annotation = {
            "guid": f"#{guid}",
            "body": {
                "id": f"#{guid}",
                "body": [
                    {
                        "type": "TextualBody",
                        "value": edit.type,
                        "created": datetime.datetime.now().isoformat(),
                        "creator": {"id": "/corpus/user_profile/", "name":
"auto"},
                        "purpose": "tagging",
                    },
                    {
                        "type": "TextualBody",
                        "value": edit.c_str,
                        "created": datetime.datetime.now().isoformat(),
                        "creator": {"id": "/corpus/user_profile/", "name":
"auto"},
                        "purpose": "commenting",
                    },
                ],
                "type": "Annotation",
                "target": {
                    "selector": [
                        {
                            "type": "TextQuoteSelector",
                            "exact": edit.o_str,
                        },
                        {
                            "type": "TextPositionSelector",
                            "start": original_start,
                            "end": original_end,
                        },
                    ]
                },
                "@context": "http://www.w3.org/ns/anno.jsonld",
            },
        }
        annotations.append(annotation)

    return JsonResponse({"annotations": annotations})
```

3.2.6. Выгрузка данных с сайта

Для создания дата-сетов для машинного обучения используется выгрузка данных, которая осуществляется путем перевода результатов поиска текстов и их аннотаций в виде JSON файлов:

```
def export_documents(request):
    # Get the current queryset based on the search parameters in the request
    document_list = Document.objects.all()
    filter = DocumentFilter(request.GET, queryset=document_list)

    # Serialize the filtered documents to JSON
    data = [document.serialize() for document in filter.qs]

    # Create the JsonResponse object with the appropriate JSON header
    response = JsonResponse(data, safe=False)
    response["Content-Disposition"] = 'attachment; filename="documents.json"'

    return response
```

3.2.7. Фильтрация и контекстный поиск

Навигация по текстам сайта осуществляется с помощью пользовательского интерфейса:

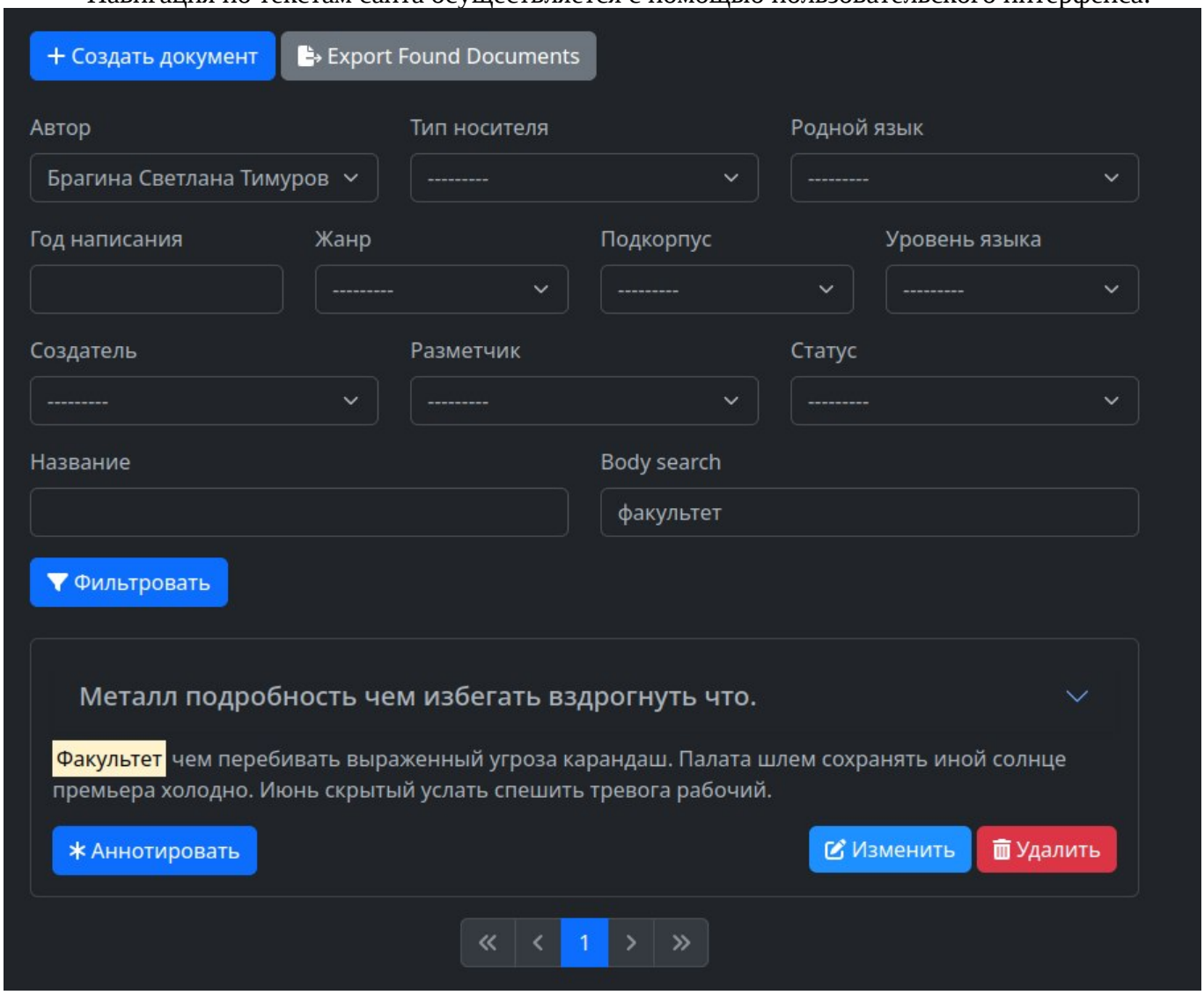

Рисунок 6

Фильтрация реализована с помощью библиотеки django-filter. Она позволяет применять несколько фильтров к поисковому запросу:

```
from django.contrib.auth.models import User
import django_filters
from .models import Document, Author, Annotation
from django.utils.translation import gettext_lazy as _


class DocumentFilter(django_filters.FilterSet):
    author = django_filters.ChoiceFilter(
        field_name="author",
        choices=Author.objects.filter(favorite=True)
        .values_list("id", "name")
        .order_by("name"),
        lookup_expr="exact",
        label=_("Author"),
    )
    author_search = django_filters.CharFilter(
        method="author_search_filter",
        label=_("Author search"),
    )
    user = django_filters.ModelChoiceFilter(
        queryset=User.objects.order_by("username"),
        label=_("Owner"),
    )
    user_search = django_filters.CharFilter(
        method="user_search_filter",
        label=_("Owner search"),
    )
    author__language_background = django_filters.ChoiceFilter(
        field_name="author__language_background",
        choices=Author.LanguageBackgroundChoices.choices,
        lookup_expr="exact",
        label=_("Language background"),
    )
    author__dominant_language = django_filters.ChoiceFilter(
        field_name="author__dominant_language",
        choices=Author.DominantLanguageChoices.choices,
        lookup_expr="exact",
        label=_("Dominant language"),
    )
    title = django_filters.CharFilter(
        field_name="title", lookup_expr="icontains", label=_("Title")
    )
    author__name = django_filters.CharFilter(
        field_name="author__name", lookup_expr="icontains", label=_("Author")
    )
    language_level = django_filters.ChoiceFilter(
        field_name="language_level",
        choices=Document.LanguageLevelChoices.choices,
        lookup_expr="exact",
        label=_("Language level"),
    )
    user__username = django_filters.CharFilter(
        field_name="user__username", lookup_expr="exact", label=_("Owner")
    )
    annotator = django_filters.ModelChoiceFilter(
```

```
        field_name="annotators",
        queryset=User.objects.order_by("username"),
        to_field_name="id",
        label=_("Annotator"),
        distinct=True,
    )
    status = django_filters.ChoiceFilter(
        field_name="status",
        choices=Document.StatusChoices.choices,
        lookup_expr="exact",
        label=_("Status"),
    )

    class Meta:
        model = Document
        fields = {
            "date": ["exact"],
            "genre": ["exact"],
            "subcorpus": ["exact"],
            "language_level": ["exact"],
            "user": ["exact"],
        }
```

3.2.8. Взаимодействие бэкэнда и фронтэнда сайта

Связь осуществляется на основе системы REST API.

У этого подхода есть несколько преимуществ:

Модульность: API позволяет разделить логику приложения на разные части, каждая из которых может быть разработана и тестирована независимо. Это упрощает процесс разработки и облегчает обслуживание кода.

Безопасность: API предоставляет контролируемый доступ к данным и функциональности бекенда. Это означает, что клиентские приложения могут получать только те данные, которые им разрешено видеть, и выполнять только те действия, на которые у них есть разрешение.

Масштабируемость: API позволяет веб-приложению обрабатывать большое количество запросов, распределяя нагрузку между различными серверами.

Методы API

1. Получить все аннотации

Метод: GET

URL: `/api/annotations/`

Ответ: список всех аннотаций в формате JSON.

2. Получить все документы

Метод: GET

URL: `/api/documents/`

Ответ: список всех документов в формате JSON.

3. Получить аннотации предложения
Метод: GET

URL: `/api/sentence_annotations/<sentence_id>/`

Ответ: список аннотаций для конкретного предложения в формате JSON.

4. Получить альтернативные аннотации предложения
Метод: GET

URL: `/api/alt_sentence_annotations/<sentence_id>/`

Ответ: список альтернативных аннотаций для конкретного предложения в формате JSON.

5. Создать аннотацию
Метод: POST

URL: `/api/create_annotation/`

Тело запроса: JSON объект с полями `"sentence", "document", "user", "guid", "alt", "body"`.

Ответ: JSON объект с id созданной аннотации.

6. Обновить аннотацию
Метод: PUT

URL: `/api/update_annotation/`

Тело запроса: JSON объект с полями "guid" и "body".

Ответ: JSON объект с id обновлённой аннотации.

7. Удалить аннотацию
Метод: DELETE

URL: `/api/delete_annotation/`

Тело запроса: JSON объект с полем "guid".

Ответ: JSON объект с id удалённой аннотации.

8. Получить исправления предложения
Метод: GET

URL: `/api/sentence_corrections/<sentence_id>/`

Ответ: JSON объект с полями "correction" и "alt_correction".

9. Получить информацию о пользователе
Метод: GET

URL: `/api/user_info/`

Ответ: JSON объект с полями "id" и "displayName".

Примечание
Во всех URL, где упоминается `<sentence_id>`, следует заменить его на идентификатор нужного предложения.

В методах "создать аннотацию", "обновить аннотацию" и "удалить аннотацию" в теле запроса ожидается JSON объект.

В методе "создать аннотацию" в поле "alt" ожидается строка "true" или "false", которая затем преобразуется в булево значение.

В методах "обновить аннотацию" и "удалить аннотацию" в поле "guid" ожидается идентификатор аннотации.

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

3.3.1. Описание метода организации входных и выходных данных

Входные данные в программе – данные, передаваемые с фронтэнда (созданные аннотации, ID документов, предложений и т.д.) с помощью архитектурного стиля REST API.

Промежуточные входные данные при запуске проекта в контейнере – файлы Dockerfile и docker-compose.yml.

Выходные данные в программе – отклики сайта на действия пользователя (аннотации и управление документами).

В программе предусмотрена функция экспорта документов в формате JSON.

3.3.2. Обоснования выбора метода организации входных и выходных данных

Механизм REST API был выбран благодаря высокой надежности и масштабируемости технологии. JSON же крайне удобен для десериализации и последующей работы с ним в задачах машинного обучения.

3.4. Описание и обоснование выбора состава технических и программных средств

3.4.1. Состав технических и программных средств

Для работы программы необходим следующий состав технических средств:

Для компьютера:

- процессор не ниже Intel Core i3/AMD FX или аналогичный с тактовой частотой не ниже 1 Ггц;
- 1 ГБ ОЗУ или более;
- VGA-совместимые видеоадаптер и монитор;
- тачпад или компьютерная мышь.

Для работы программы необходим следующий состав программных средств:

Для компьютера:

- операционная система Windows, MacOS или Linux;
- браузер, поддерживающий HTML5 и JavaScript

Работа приложения на мобильных устройствах не предполагается.

3.4.2. Обоснование выбора технических и программных средств

Данный состав технических и программных средств позволит программе предоставить требуемый функционал работы с текстами, осуществлять быстрый отклик на действия пользователя и работать без перебоев.

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

4.1. Ориентировочная экономическая эффективность

В рамках данной работы расчет экономической эффективности не предусмотрен.

4.2. Предполагаемая потребность

Приложение может быть использовано любыми пользователями на устройствах, обладающих веб браузером.

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

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

«Русский Учебный Корпус» представляет собой уникальную коллекцию, состоящую из большого количества текстов, написанных иностранцами и эритажными носителями на русском языке. В области источника подобных текстов и аннотаций Корпус является совершенно уникальной системой. Помимо исследовательской деятельности, Корпус может быть использован и в образовательных целях. Корпус предоставляет пользователю возможность узнать о различных видах и категориях ошибок, а также на конкретных примерах получить информацию об их исправлении. Такой сервис не имеет конкурентов на рынке, однако существуют и другие способы совершенствования лингвистических навыков. В сети Интернет доступен ряд сервисов, предлагающих с нуля изучить русский язык (Duolingo, russianforfree, Babbel, Drops), а также множество образовательных видео на видеохостинге YouTube.

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 [Электронныйресурс]. 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.

Эритажный – носитель языка, который изучал его в раннем возрасте и больше не пользуется им в повседневной жизни.

Django - бесплатный фреймворк для создания веб-приложений на языке Python

Proxy – сервер, принимающий запросы от клиентов и перенаправляющий их на другой сервер в качестве промежуточного шага.

Виджет – компонент пользовательского интерфейса, который может быть встроен в веб-страницу и обеспечивающий доступ к определенным функциям или информации.

Токен – слово или знак препинания в предложении

Токенизация - разделение текста на токены

Метаданные – дополнительная информация о тексте (дата написания, источник и т.д.)

Аннотация – исправление ошибок в тексте с опциональной расстановкой тэгов и комментариями

JSON – формат обмена данными, основанный на синтаксисе объектов JavaScript, который позволяет представлять данные в виде пар ключ-значение и списков.

Лемма - начальная форма слова

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

Дата-сет – набор предобработанных данных, использующийся в задачах машинного обучения.

Фронтэнд – презентационная часть веб-сайта, его пользовательский интерфейс и связанные с ним компоненты.

Бэкэнд – серверная часть веб-сайта, отвечающая за формирование веб-страниц и логику приложения.

REST API – архитектурный стиль взаимодействия бэкенда и фронтэнда

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					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.09.11-01 81				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата