Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка классического статического сервера для отдачи контента с диска»

Студент группы **ИУ7-75Б** _____ **П. А. Иванов**
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ **Н.О. Яковидис**
(Подпись, дата) (И.О. Фамилия)

***2023** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 84 с., 24 рис., 4 табл., 44 ист., 2 прил.

ПАРАЛЛЕЛЬНЫЙ КОРПУС, РАЗМЕТКА ПАРАЛЛЕЛЬНОГО КОРПУСА, БАЗЫ ДАННЫХ, РЕЛЯЦИОННЫЕ СУБД, КЕШИРОВАНИЕ, ЛОГИРОВАНИЕ

Объектом исследования данной работы является корпус параллельных текстов.

Целью работы является разработка базы данных для хранения и обработки параллельного корпуса переведённых текстов.

Поставленная цель достигается путем анализа предметной области параллельных корпусов текстов и существующих систем управления базами данных, проектированием и реализацией базы данных для хранения и пополнения параллельного корпуса. Кроме того, реализована система кеширования и исследовано влияние её использования на время исполнения запросов к системе.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Лексема – словарная единица, рассматриваемая как совокупность форм и значений, свойственных одному и тому же слову во всех его употреблениях и реализациях [1].

Словоформа – одна из косвенных форм слова, получаемая путем спряжения либо склонения исходного слова [1].

БД – база данных [19].

СУБД – система управления базами данных [19].

ВВЕДЕНИЕ

Параллельный корпус – неотъемлемая часть науки о переводе. В наши дни параллельный корпус стал неотъемлемой частью уже и самого процесса перевода. Переводчику необходимы ресурсы, предлагающие ему варианты интерпретации исходного текста для подтверждения собственных гипотез. На практике больше половины времени, затрачиваемого на перевод, уходит на просмотр вспомогательных материалов [2]. Таким образом, использование параллельного корпуса может существенно ускорить работу переводчика.

Целью данной курсовой работы является разработка базы данных для хранения и обработки параллельного корпуса переведённых текстов.

Для достижения поставленной цели необходимо решить следующие задачи:

— проанализировать предметную область параллельных корпусов текстов, сформировать требования для хранения разметки текстов;
— проанализировать существующие СУБД;
— спроектировать БД для хранения и пополнения параллельного корпуса текстов и ролевую модель;
— спроектировать кеширование запросов с использованием дополнительной БД, спроектировать систему логирования действий пользователей;
— выбрать средства реализации системы, реализовать спроектированную БД и необходимый интерфейс для взаимодействия с ней;
— реализовать систему логирования действий пользователей и провести тестирование разделения ролей;
— провести нагрузочное тестирование, определить зависимость времени исполнения запросов от использования кеширующей БД.

1 Аналитический раздел

1.1 Определение параллельного корпуса

Под «лингвистическим корпусом» понимают достаточно объемное собрание примеров языкового массива, определенным образом структурированное, унифицированное и размеченное, объединенное общим логическим замыслом (доминантой), хранящееся в электронной форме, позволяющее решать лингвистические задачи. Понятие «корпуса текстов» близко к понятию лингвистического корпуса и представляет собой более частную форму проявления последнего. В случае корпуса текстов, его доминантой является одна или совокупность определенных целей: обучение иностранному языку, отладка систем машинного перевода и прочие [3].

Корпус, содержащий тексты только на одном языке, называется монолингвистическим (англ. «monolingual corpus»). В случае хранения текстов на двух языках, корпус называют билингвистическим (англ. «bilingual corpus»).

Параллельный корпус – билингвистический корпус, содержащий тексты одновременно как на его изначальном языке, так и на некотором другом языке [4].

1.2 Применение параллельного корпуса

Параллельный корпус является ценным ресурсом для исследователей самых разных сфер, таких как машинный перевод, перевод с помощью компьютера, изучение языков с помощью электронных ресурсов, переводоведение, контрастивная лингвистика [5]. Рассмотрим подробнее основные области применения параллельного корпуса: машинный перевод, инструменты обработки естественного языка, языкознание и работа переводчика или преподавателя.

1.2.1 Машинный перевод

Параллельный корпус является важнейшей частью работы статистических систем машинного перевода, а также систем, работа которых основана на правилах. Данные системы постоянно анализируют доступные тексты для вычисления наиболее вероятного перевода. Кроме того, параллельный корпус используется и в методе машинного перевода с использованием глубокого обучения.

1.2.2 Инструменты обработки естественного языка

Некоторые инструменты NLP (аббр. Natural Language Processing – обработка естественного языка) также зависят от параллельного корпуса. Примерами таких инструментов являются программы для кросс-языкового извлечения данных (англ. «Cross Language Information Retrieval»), транслитерации (побуквенной передачи отдельных слов и текстов одной графической системы средствами другой системы), систем ответов на вопросы (англ. «Question Answering Systems»), предназначенных для ответов на вопросы на естественном языке, и другие.

1.2.3 Языкознание

Одно слово в языке может иметь множество значений в зависимости от контекста [12]. Таким образом, параллельный корпус помогает в исследованиях влияния окружения слова на его перевод (при наличии достаточного количества загруженных текстов). С помощью корпуса выявляются характерные для языка паттерны использования конкретных слов, фраз. Кроме того, на основе текстов

делаются выводы об особенностях морфологии, синтаксиса языков, а также вытекающих из них особенностях перевода. Самым ярким примером науки, которой необходим параллельный корпус, является контрастивная лингвистика.

1.2.4 Работа переводчика или преподавателя

В связи с множеством возможных переводов слова или фразы в зависимости от контекста, работа переводчика неразрывно связана с использованием лингвистических инструментов. Как правило, 50% времени, затрачиваемого на перевод – это использование справочных материалов [2]. Использование параллельного корпуса призвано облегчить работу переводчика, поскольку корпус содержит множество примеров перевода слов, фраз и предложений в зависимости от контекста их употребления. Кроме того, корпус может помочь людям, изучающим иностранный язык.

1.3 Требования к корпусу

При создании лингвистического корпуса рекомендуется придерживаться следующих правил [6].

1) Тексты корпуса должны выбираться без учёта специфики языков, на котором они написаны.
2) Содержание корпуса должно быть репрезентативным.
3) Только те компоненты корпуса, которые спроектированны быть независимо контрастивными, должны быть контрастивными.
4) Вся информация о тексте должна храниться отдельно, но может быть объединена при необходимости.
5) Экземпляры текстов, включаемые в корпус, должны добавляться туда це-

ликом (или настолько в целостном состоянии, насколько это возможно).

6) Состав и структура корпуса должны быть задокументированы (иначе корпус бесполезен).
7) Разработчик должен стремиться к созданию разнообразного, сбалансированного корпуса.
8) Тексты должны быть однородными (гомогенными), следует избегать текстов плохого качества.

Важно отметить, что эти рекомендации важны не столько в терминах практических советов, сколько как указание на важность учета теории при разработке корпуса.

Многие авторы отмечают, что не существует «идеального» корпуса. Как указал М. Нельсон, каждая попытка создать корпус суть компромис между ожидаемым и достигаемым [7]. Тем не менее, при попытке создать собственный корпус важно учитывать вышеперечисленные рекомендации.

Что касается размера корпуса, важно учитывать его предназначение. Если параллельный корпус требуется для исследования редкоупотребляемых слов, то требуется корпус большого размера, т.е. имеющий большое количество текстов разных авторов из разных жанров. В случае обыкновенного перевода или исследования базовых слов языка, такое разнообразие не требуется. Таким образом, размер корпуса – открытый вопрос, который решается для каждого корпуса в отдельности.

Репрезентативность корпуса означает, что корпус учитывает лингвистическое разнообразие данного языка и выбранной темы для исследования. Смежное качество – сбалансированность в отношении структуры корпуса и данных для его заполнения. Утверждается, что хорошо сбалансированный корпус содержит несколько подразделов, представляющих разные варианты использования языка (наличие текстов разных жанров). Причем важно, чтобы разделы имели примерно одинаковое число слов [6].

1.4 Работа с корпусом

Рассмотрим этапы работы с корпусом [5]:

1) сбор материала для корпуса;
2) выравнивание;
3) разметка;
4) использование корпуса.

Источником данных могут выступать политические документы, инструкции и документации к ПО, переведённые веб-сайты. Примерами таких источников можно назвать сайт Европарламента, сайты дипломатеческих представительств или открытую информацию с сайта gnu.org.

Главная задача выравнивания – способствование поиску переводов. В то время как в монолингвистическом корпусе поиск производится с целью извлечения примеров перевода, в параллельном корпусе результатом поиска должны быть и переводы соответствующих примеров употребления. В результате выравнивания имеется некоторое соответствие между предложениями (в худшем случае, абзацами и главами) на двух языках.

Разметка добавляет дополнительную информацию в корпус, которая и является преимуществом корпуса над классическими словарями или сборниками текстов. Выделяют несколько видов разметки. Выбор вида разметки зависит от целей исследований, для которых используется корпус.

Выровненный, размеченный корпус является готовым инструментом для исследований. Он может служить помощником как переводчику или преподавателю, так и исследователю-лингвисту. Основной операцией в корпусе является поиск с указанными параметрами. Причем допустимые параметры могут варьироваться от корпуса к корпусу.

1.5 Разметка параллельного корпуса

Именно разметка отличает параллельный корпус от обычного собрания текстов. Зачастую разметку также называют аннотацией. Выделяются следующие основные виды (уровни) разметки [8][9]:

— просодическая;
— морфологическая (морфосинтаксическая);
— синтаксическая (грамматическая);
— семантическая;
— прагматическая;
— разметка дискурса;
— анафорическая (местоименная);
— метаразметка (параметры текстов).

Наличие конкретного вида разметки во многом определяет назначение корпуса. Если необходим анализ употребления конкретных грамматических форм, требуется грамматическая разметка.

Кроме того, разметка может помочь в поиске точных примеров использования конкретного слова. Если, например, необходимо получить все примеры употребления напитков (или предметов одежды) в текстах, необходима семантическая разметка.

Просодическая разметка используется для фиксации ударений в словах и интонации в случае их произношения. Она применяется, как правило, в корпусах устной разговорной речи и сопровождается дискурсной разметкой, которая указывает на наличие пауз, повторов или оговорок. Данные виды разметки сложны для автоматизации и проводятся вручную.

Анафорическая разметка фиксирует референтные связи (например, местоименные). Однако, поскольку большая часть программ анализирует тексты, предварительно разбив их на предложения, и, следовательно, связи между пред-

ложениями теряются, анафорическая разметка проводится вручную.

Прагматическая разметка применяется для указания прагматических функций фразы, предложения или абзаца. Прагматическая функция лингвистической единицы зависит от дискурса, личности повествователя и др., и показывает, как связаны повествователь, его фразы и действия, которые он совершает. Автоматизация прагматической разметки также затруднена [10].

Аннотация текста может быть произведена несколькими способами. Во-первых, она может быть проведена полностью вручную. Во-вторых, возможен вариант ручной разметки с помощью электронных средств. Наконец, существует полностью автоматический вариант разметки. Примером разметки, которая может быть проведена полностью автоматически, является морфологическая разметка [6].

1.5.1 Морфологическая разметка

Определение морфологических форм и значений является самостоятельной научной проблемой. Морфологическая информация, приписываемая произвольному слову в тексте, может состоять из следующих полей [8]:

— лексема, которой принадлежит словоформа – указывается часть речи;
— множество грамматических признаков лексемы, словоклассифицирующие характеристики (род для существительного, переходность для глагола и т.д.);
— множество грамматических признаков словоформы, или словоизменительные характеристики (падеж для существительного, число для глагола);
— информация о нестандартности грамматической формы.

Например, в Национальном корпусе русского языка (НКРЯ) [11] используется следующая морфологическая разметка на основе латинских букв [8].

1) Части речи:

— *S* – существительное,

— *A* – прилагательное,

— *NUM* – числительное,

— *V* – глагол,

— *ADV* – наречие,

— *PARENTH* – вводное слово.

2) Значения грамматических категорий [1]:

— *m* – мужской род,

— *f* – женский род,

— *m-f* – «общий» род,

— *n* – средний род.

3) Одушевленность:

— *anim* – одушевленный,

— *inan* – неодушевленный.

1.5.2 Метаразметка

Под метаразметкой понимается приписывание тексту атрибутов, отражающих обстоятельства его создания, автора, тематику, жанровые особенности и другие. Цель метаразметки – разграничить тексты разного качества, разных жанров, разной даты написания для получения наиболее релевантных ответов относительно запроса [8].

Частые атрибуты для указания параметров текста:

— автор текста,

— время создания текста,

[1] Стоит отметить, что есть языки, в которых отсутствует понятие рода. Пример – турецкий язык. Нет разницы между «братом» и «сестрой» – это все «kardeş». Таким образом, не все перечисленные морфологические признаки могут быть выявлены для слов конкретной языковой пары.

— объем текста,

— название текста,

— жанр (тип) текста,

— тематика текста.

В случае подхода к наполнению корпуса с помощью интернета (т. н. Web Mining Approach), логично также указывать ссылку на источник, режим, время и дату доступа [4].

1.5.3 Синтаксическая разметка

В случае синтаксической разметки, каждому предложению текста приписывается его синтаксическая структура. Существует множество моделей синтаксической структуры предложения. Например, НКРЯ использует лингвистическую модель И.А. Мельчука «Смысл ⇔ текст». В данной модели синтаксическая структура представляет собой дерево зависимостей, в узлах которого стоят слова предложения, а дуги помечены именами синтаксических отношений [8].

Пример синтаксического дерева для предложения «Виктор сдал экзамен» представлен на рисунке 1.

Синтаксическая разметка – сложная задача, которая для каждого языка решается индивидуально. Более того, синтаксическая разметка плохо применима в средствах машинного перевода [12]. Её использование оправдано при проведении специальных исследований.

1.5.4 Семантическая разметка

При данной разметке большинству слов в тексте приписываются один или несколько семантических или словообразовательных признаков. Примера-

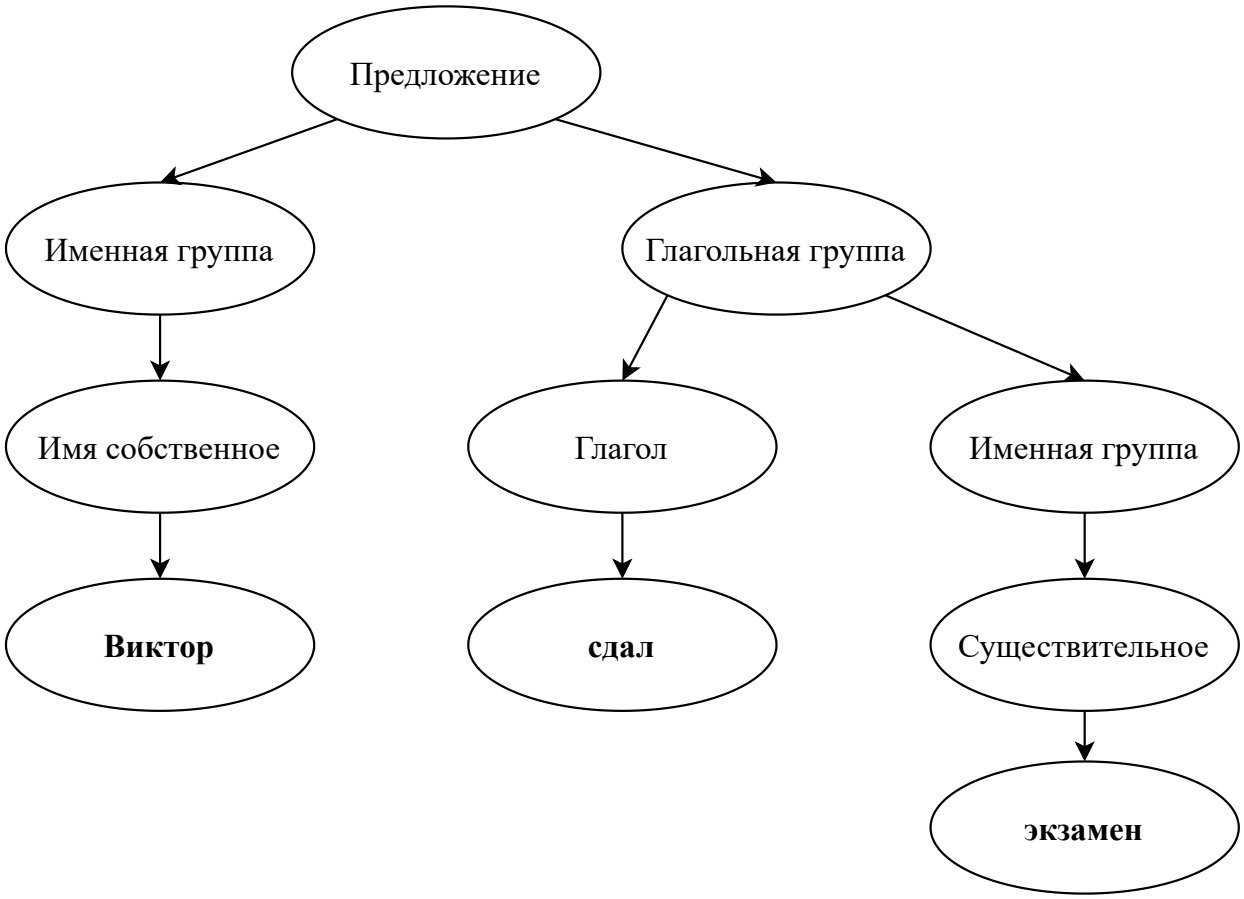

Рисунок 1 – Пример синтаксического дерева

ми признаков могут быть движение, вещество, одежда, свойство человека и т.д. При этом одно слово может попадать в несколько классов.

Семантическая разметка – крайне трудная задача из-за проблемы многозначности слов. Трудно отнести слово, которое имеет десятки значений при разном контексте, к конкретному выделенному классу [8].

1.6 Требования к базе данных и хранимой информации

В соответствии с описанной спецификой хранения данных в параллельном корпусе текстов, разработанная база данных должна соответствовать следующим требованиям.

— В базу данных загружаются тексты в текстовом формате с минимальной,

заранее описанной аннотацией (метаразметкой текста).

— После загрузки текста должно быть произведено выравнивание текста по предложениям, затем предложений по словам.
— После выравнивания по словам, должна быть произведена морфологическая аннотация каждого слова.
— Должна быть возможность быстрого поиска по базе данных для получения перевода слова, примеров употребления слова в предложениях с возможностью указать параметры разметки: морфологические признаки слова или изменить параметры выборки текстов.
— Система должна быть расширяема до любого количества языков.
— Должна быть предоставлена возможность совместного использования корпуса.

Поскольку семантическая и синтаксическая аннотация сложны и используются чаще всего только при специализированных исследованиях, их наличие от корпуса не требуется. К морфологической и метаразметке предъявляются следующие требования:

— по каждому слову (кроме его перевода) текстов должна быть информация о его исходном языке, части речи, грамматической категории и одушевленности;
— о каждом тексте должна быть известна информация о его авторе, времени доступа к информации, объему текста (в предложениях), названии и его жанре. В случае онлайн источника, должна быть указана ссылка на источник.

1.7 Существующие решения

Как отмечается, не существует универсальной архитектуры для построения корпуса [4]. Большинство существующих решений используют XML (eX-

tensible Markup Language) [13], HTML (HyperText Markup Language) [14] и TXT файлы для хранения корпуса. В редких случаях используют Excel файлы [4]. В последнее время стали появляться корпусы на основе реляционных баз данных [16] [15].

При этом стоит отметить, что самое распространенное решение – XML файлы – обладает рядом недостатков [17].

— Синтаксис XML избыточен;
— Долговременное хранение XML-документа нерационально: его размер существенно больше бинарного представления тех же данных;
— Избыточность XML может повлиять на эффективность приложения. Возрастает стоимость хранения, обработки и передачи данных;
— Скорость поиска по корпусу будет равна скорости поиска по текстову документу, заполненного, кроме основного текста, большим числом тегов разметки.

Однако XML данные подходят для загрузки данных в базу для их дальнейшей обработки вне этого файла.

Пример организации загрузки XML файла представлен на рисунке 2 [17].

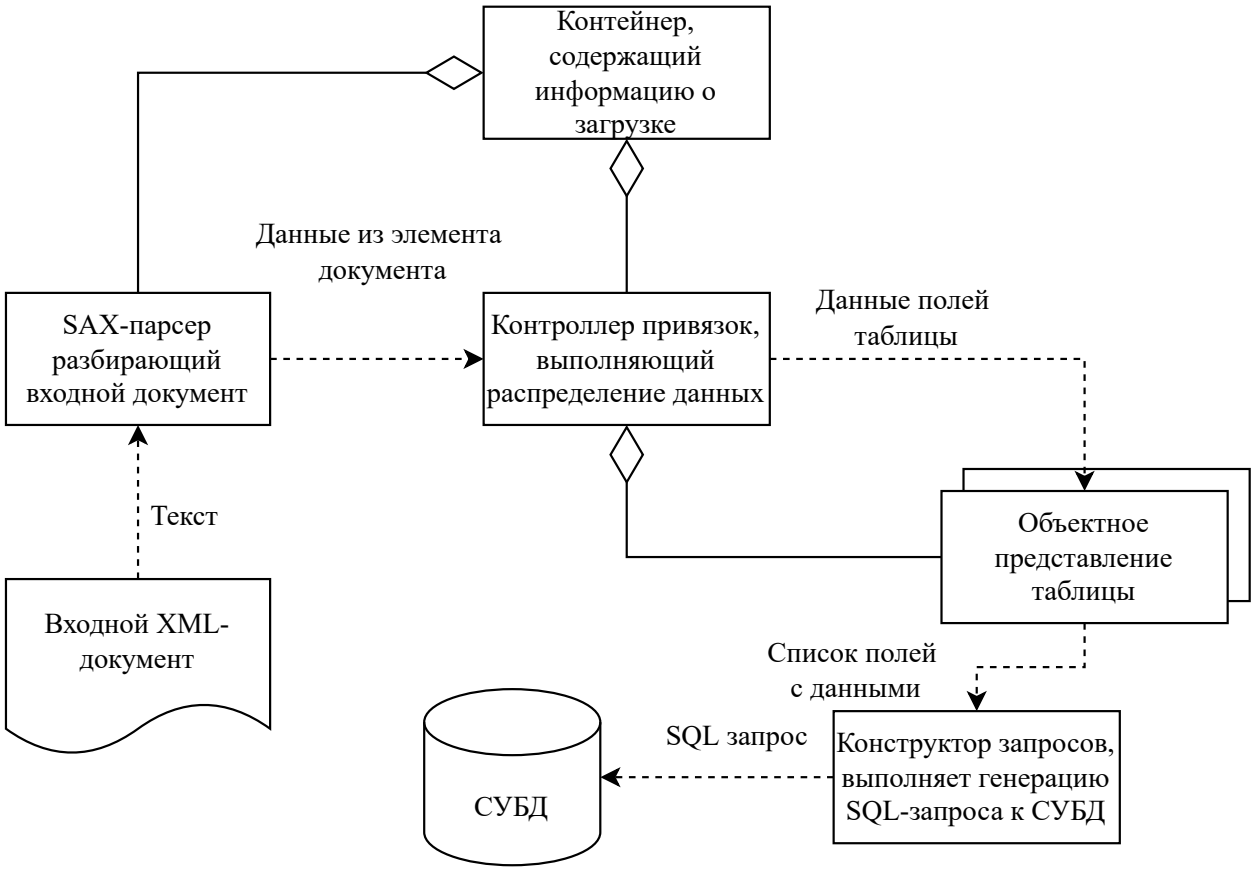

Рисунок 2 – Пример организации загрузки XML документов в СУБД

Что касается примеров существующих решений, выделяются следующие продукты: Reverso [24], Abbyy Lingvo [25] и Linguee [26]. Преимущества и недостатки этих продуктов представлены в таблице 1.

Таблица 1 – Сравнение существующих решений

Продукт	Преимущества	Недостатки
Reverso	— поддержка; — большая база текстов.	— платная подписка; — нельзя фильтровать тексты; — нельзя загружать тексты.
Abbyy Lingvo	— возможность добавления собственных материалов.	—прекращена поддержка; — ориентированность на словари вместо текстов; — нельзя фильтровать тексты.
Linguee	— поддержка; — большая база текстов; — наличие как словаря, так и переводчика.	— нельзя фильтровать тексты.

Из приведенного анализа видно, что общий недостаток существующих решений – отсутствие возможности самостоятельной загрузки текстов или хотя бы поиска по определенной выборке текстов. Также недостатком является необходимость платной подписки для получения полного функционала в некоторых решениях.

1.8 Анализ существующих СУБД

Рассмотрим существующие системы управления базами данных, но сначала определим основные понятия.

База данных (БД) – это самодокументированное собрание интегрированных записей [18].

1) БД является самодокументированной, т.е. она содержит описание собственной структуры. Это описание называется словарем данных, катало-

гом данных или метаданными.

2) БД – это собрание интегрированных записей, она содержит:
   — файлы данных,
   — метаданные,
   — индексы,
   — метаданные приложений.
3) БД является информационной моделью пользовательской модели предметной области.

Выделяют следующие требования к архитектуре базы данных [20].

— Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя свое собственное представление о них. Каждый пользователь должен иметь возможность изменить свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей.
— Пользователи не должны непосредственно иметь дело с подробностями физического хранения данных в базе.
— Администратор базы данных должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.
— Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации, как переход на новое устройство хранения.
— Администратор базы данных должен иметь возможность изменять концептуальную или глобальную структуру базы данных без какого-либо влияния на всех пользователей.

Система управления базами данных (СУБД) – это совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия её с прикладными программами. При этом СУБД должна обеспечивать совместное использование

данных многими пользователями, безопасность данных, эффективный интерфейс доступа к данным [19].

Классификация СУБД проводится по нескольким признакам. Рассмотрим классификацию по модели данных и по архитектуре организации хранения данных [19] [20].

1.8.1 Модели данных

Выделяют следующие СУБД по используемой ими модели данных [20].

- Дореляционные:
  - инвертированные списки (файлы),
  - иерархические,
  - сетевые.
- Реляционные.
- Постреляционные.

Иерархическая модель может быть представлена как древовидный граф с записями в виде узлов и множествами в виде ребер. Схема иерархической архитектуры представлена на рисунке 3.

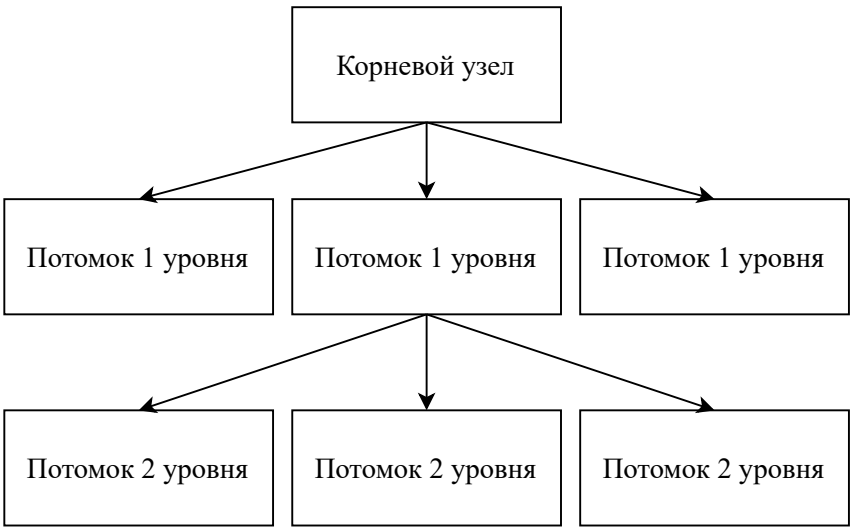

Рисунок 3 – Схема иерархической архитектуры

В иерархической модели узел может иметь только одного родителя. Узлы представляют собой интересующие нас объекты, а связи между ними определяются указателями, содержащими в себе информацию о физическом расположении данных [20]. Структура узла представлена на рисунке 4.

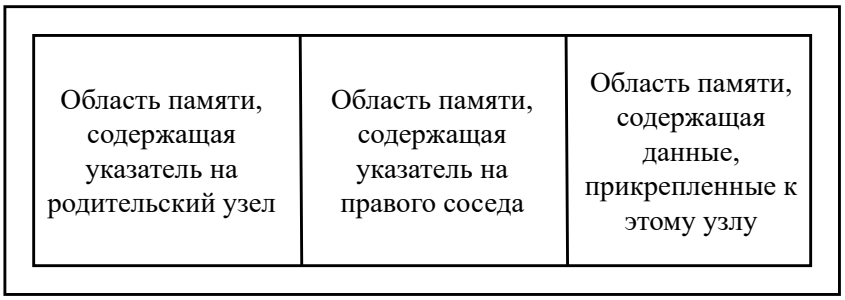

Рисунок 4 – Структура узла

Сетевой подход является развитием иерархической архитектуры: потомок может иметь любое число предков. В сетевой модели логика процедуры выборки данных зависит от физической организации этих данных. Таким образом, модель не является полностью независимой от приложения: если необходимо изменить структуру данных, то придется модифицировать и приложение [20]. Пример сетевой модели представлен на рисунке 5.

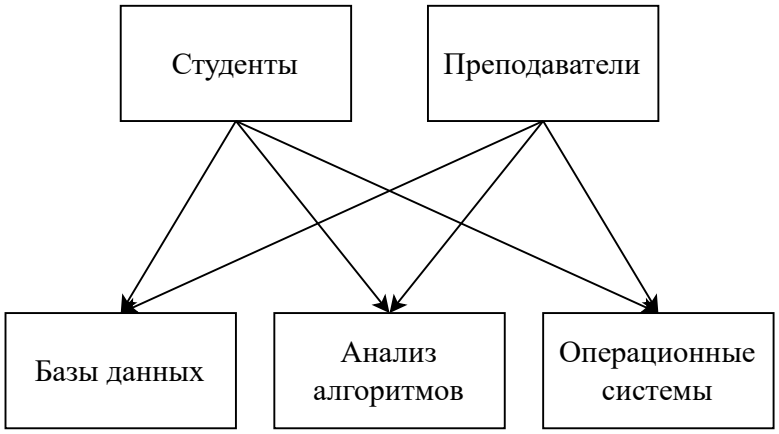

Рисунок 5 – Пример сетевой модели данных

СУБД на основе инвертированных списков представляет собой совокупность инвертированных файлов, отличающихся простотой организации и наличием весьма удобных языков манипулирования данными [20]. База данных на инвертированных списках состоит из таблиц отношений, однако она сильно отличается от реляционной модели. В такой модели данных отсутствуют ограничения целостности, допускается сложная структура атрибутов. Все ограничения на возможные экземпляры БД задаются теми программами, которые работают с БД. Пользователь может управлять логическим порядком строк в каждой таблице с помощью специального инструмента – индексов. Эти индексы автоматически поддерживаются системой и явно видны пользователям [21]. Пример организации инвертированных списков представлен на рисунке 6.

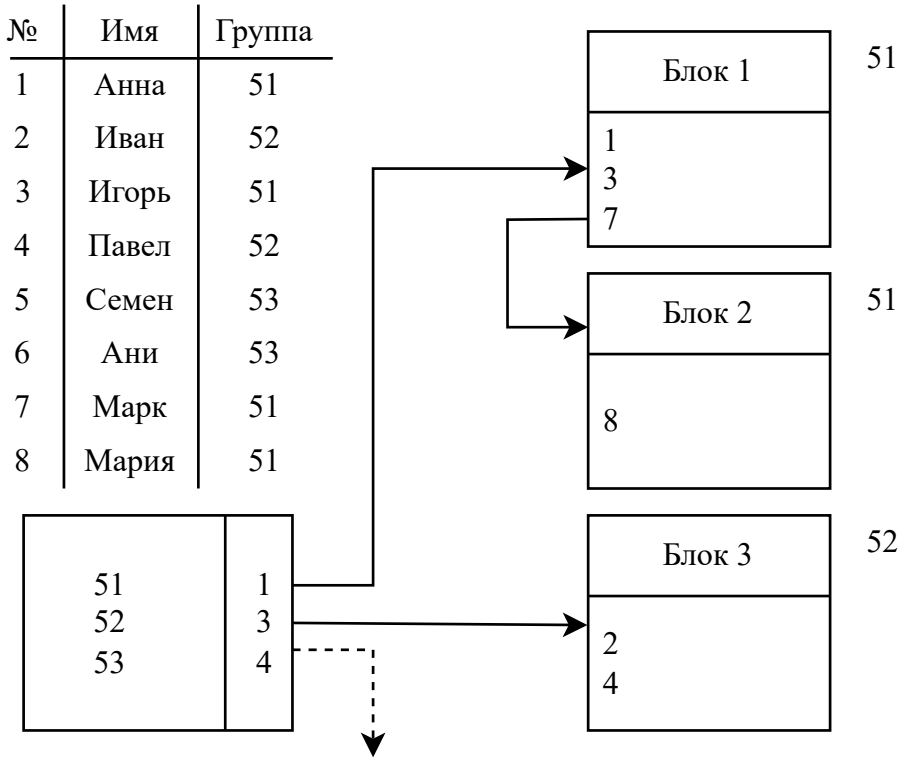

Рисунок 6 – Пример организации инвертированных списков

Реляционная модель данных включает следующие компоненты [21]:

— структурный,

— целостный,

— манипуляционный.

В структурной части модели фиксируется, что единственной структурой данных, используемой в реляционных базах данных, является нормализованное n-арное отношение. В манипуляционной части модели выделяются два фундаментальных механизма управления реляционными базами данных: реляционная алгебра и реляционное исчисление. В целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД [21].

Кроме того, в состав реляционной модели данных включают теорию нормализации. Нормализация осуществляется для уменьшения числа возможных аномалий (нежелательные последствия при чтении, вставке, изменении или удалении данных) посредством приведения отношений к нормальной форме (НФ).

Выделяют 1НФ, 2НФ, 3НФ, 4НФ, 5НФ, НФБК (нормальная форма Бойса-Кодда) [18]. Однако отмечается, что зачастую нет необходимости в использовании нормальных форм четвертого порядка и выше [22]. Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и не имеет транзитивных зависимостей [18].

В рамках постреляционной модели выделяют распределенные файловые системы, NoSQL (Not Only SQL) СУБД, объектные, объектно-реляционные базы данных. Отмечается, что наилучшим решением для корпоративной информационной системы оказались многопользовательские централизованные и распределенные базы на основе строго типизированной реляционной модели с транзакционной обработкой данных. Однако в системах, где, например, возможно создание новых моделей данных, не требующих строго фиксированной структуры, или постоянное расширение круга пользователей, используются именно постреляционные СУБД [23].

1.8.2 Архитектура организации хранения данных

В данной классификации выделяют централизованные СУБД, которые работают с БД, которая физически хранится в одном месте (на одном компьютере). Это не означает, что пользователь может работать с БД только за этим же компьютером: доступ может быть удаленным, в режиме клиент–сервер. Кроме того, выделяются распределенные СУБД: части системы могут размещаться на двух и более компьютерах [19].

1.9 Сущности проектируемой базы данных

На основе выявленных требований к базе данных и хранимой информации уместно выделить следующие сущности: текст, предложение и слово.

Более подробное разбиение текста на разделы, главы, пункты (или другие структурные единицы текста) нецелесообразно, поскольку в зависимости от жанра их наличие может варьироваться. Что касается дополнительной декомпозиции предложений, хранение фраз или словосочетаний также нецелесообразно, поскольку в условиях расширяемости до любого количества языков трудно предусмотреть особенности группировки слов абсолютно каждого языка. Кроме того, затруднено выравнивание фраз в предложениях.

Что касается разметки, уместно выделить каждый вид разметки в отдельную сущность для возможности последующего расширения. В базу данных будем добавлять морфологическую аннотацию и метаразметку текстов. Синтаксическая и семантическая разметка не будут использоваться, так как они используются в основном при специализированных исследованиях.

Кроме того, выделим такие сущности как «язык», «языковая пара» для разграничения текстов на разных языках.

В связи с необходимостью обеспечения возможности совместного использования корпуса, необходимо также хранить минимальную информацию о пользователях: фамилию, имя, почту для обратной связи и страну, выделенную в отдельную сущность. Хранение страны необходимо для возможного последующего подсчета статистики по использованию корпуса.

Диаграмма «сущность-связь» в нотации Чена представлена на рисунке 7.

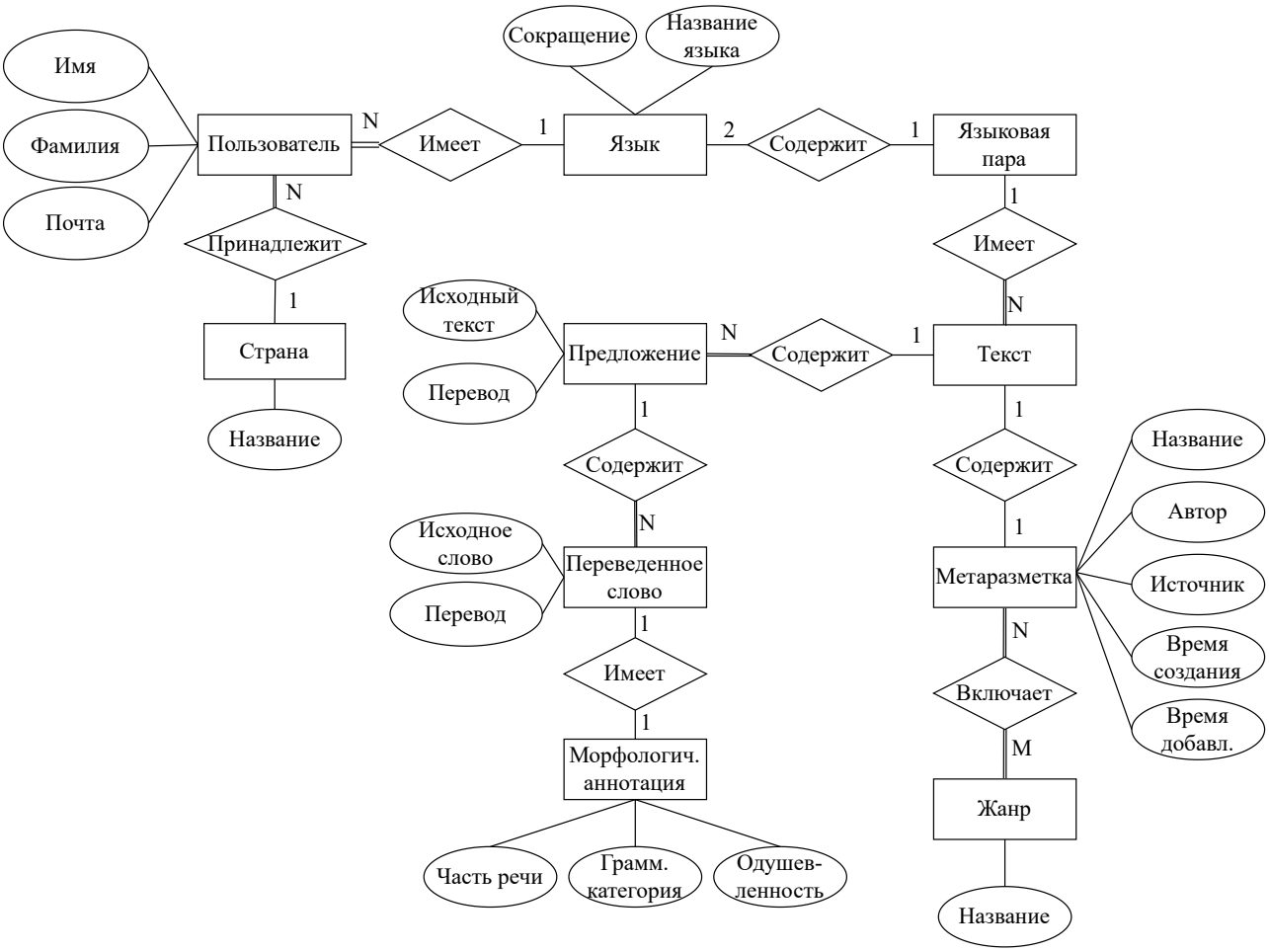

Рисунок 7 – ER-диаграмма базы данных

1.10 Описание пользователей проектируемого приложения к базе данных

Чтобы формализовать пользователей, необходимо четко сформулировать цель корпуса. В данном случае, это помощь в работе переводчикам и преподавателям иностранных языков. Таким образом, выделяются роли переводчика и преподавателя. Их различие состоит в отсутствии необходимости загрузки новых переведенных текстов в систему у преподавателя. Преподаватель занимается поиском контекста употребления слов, когда как переводчик может загружать новые тексты (например, из собственной практики). Также уместно выделить роль администратора, который будет управлять пользователями, и неав-

торизованного пользователя, который впоследствии станет переводчиком или преподавателем.

Диаграмма вариантов использования представлена на рисунке 8.

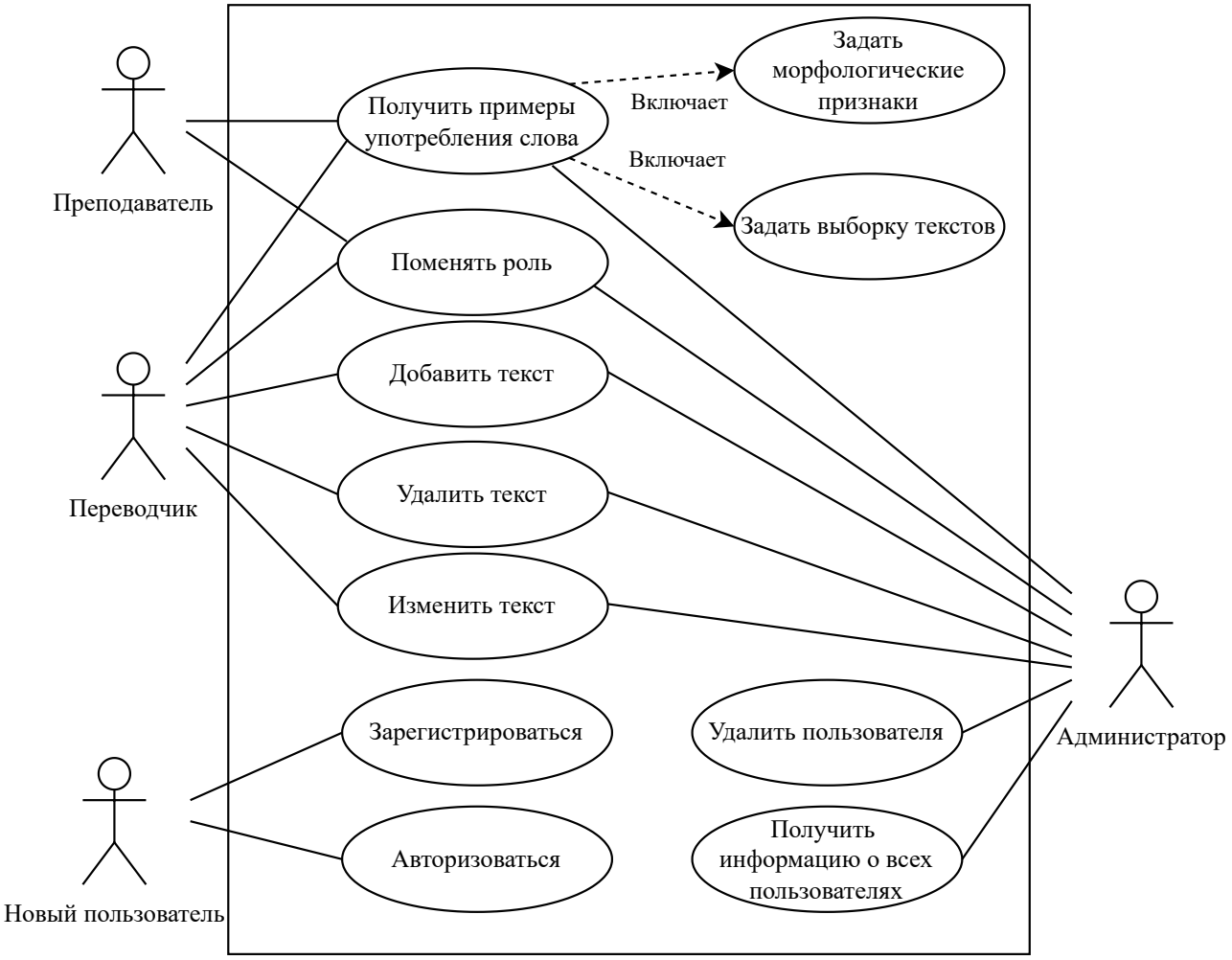

Рисунок 8 – Диаграмма вариантов использования

Для поддержания качества загружаемых текстов, администратор должен иметь возможность удалять и изменять загруженные тексты, а также удалять недобросовестных пользователей. Переводчик также может изменять и удалять тексты, но только самостоятельно загруженные. И преподаватель, и переводчик должны иметь возможность получить примеры употребления заданного слова, возможно, с указанием необходимых морфологических признаков и требуемой категории текстов.

Вывод

В данном разделе была рассмотрена предметная область корпусов переведенных текстов: были даны определения ключевым понятиям области параллельных корпусов, рассмотрены варианты использования корпуса. Кроме того, были сформулированы требования к наполнению корпуса и его возможностям.

Из рассмотренных видов разметки для разрабатываемой базы данных выбрана морфологическая аннотация и метаразметка. Использование синтаксической или семантической разметки нецелесообразно, поскольку данные виды разметки используются, как правило, лишь при специализированных исследованиях.

Были также рассмотрены существующие решения и выявлен их главный недостаток – отсутствие возможности фильтрации текстов по тематике для получения наиболее релевантных результатов. Кроме того, зачастую готовые решения хранят данные в устаревшем формате XML и лишь недавно начали появляться продукты, использующие современные разработки в области баз данных.

Из анализа существующих решений можно сделать вывод, что наиболее подходящим для решения поставленной задачи видом СУБД является реляционная база данных, развернутая локально (централизованная СУБД).

На основе выявленных требований к базе данных и хранимой информации выделены сущности «текст», «предложение», «слово», «предложение» и другие. Были также разработаны и представлены на диаграмме «сущность-связь» связи между сущностями.

Наконец, были формализованы пользователи приложения к базе данных и их возможности, которые были представлены на диаграмме вариантов использования.

2 Конструкторский раздел

2.1 Проектирование базы данных

В соответствии с диаграммой «сущность-связь», представленной на рисунке 7, база данных должна содержать следующие таблицы:

— таблица текстов text;
— таблица жанров текстов genre;
— таблица для связи метаразметки и жанров meta_genre;
— таблица метаразметки meta_annotation;
— таблица языковых пар language_pair;
— таблица предложений sentence;
— таблица слов word;
— таблица морфологической аннотации morphological_annotation;
— таблица пользователей user;
— таблица стран country;
— таблица языков language.

Диаграмма проектируемой базы данных представлена на рисунке 9.

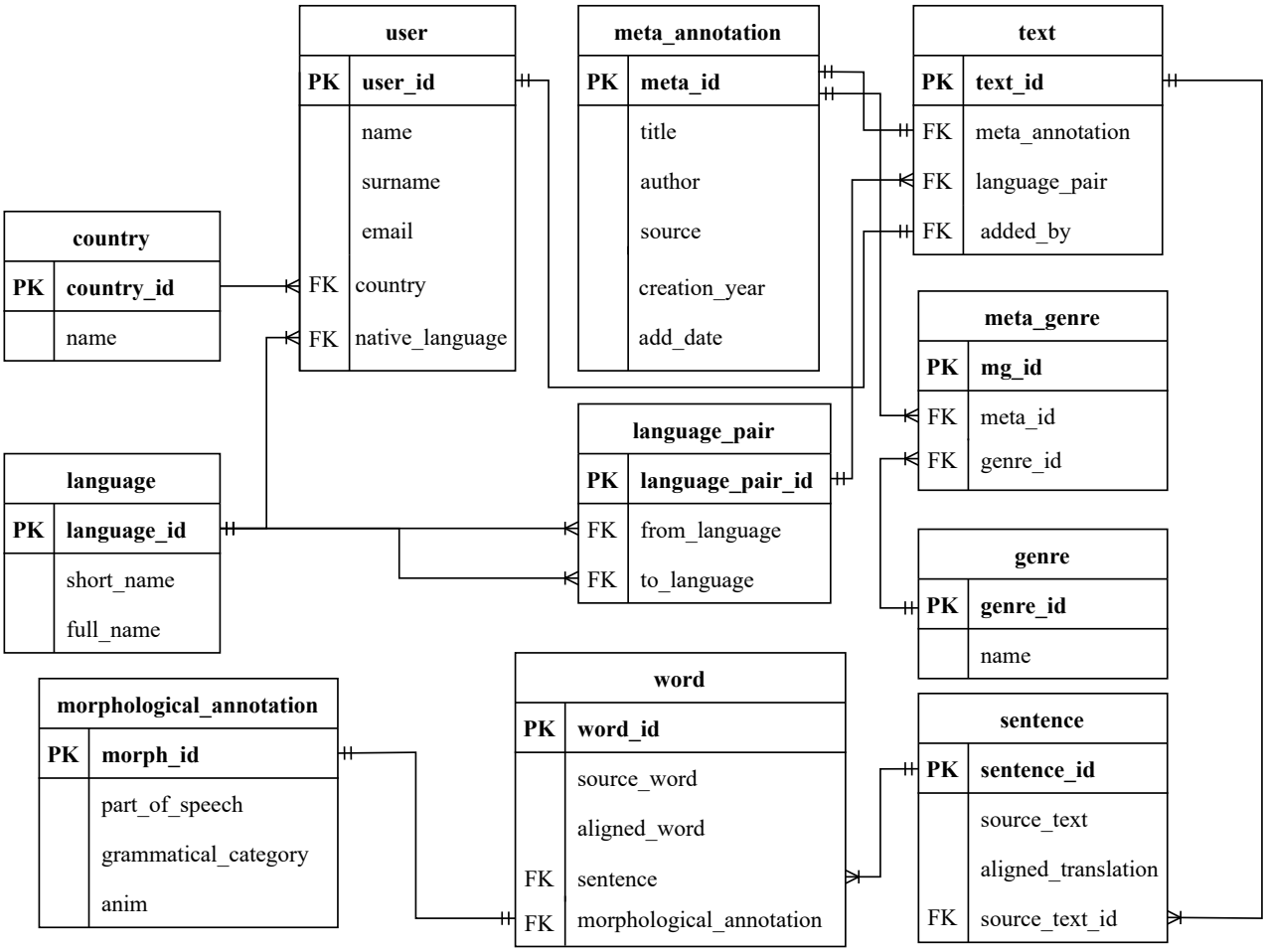

Рисунок 9 – Диаграмма БД

Таблица meta_annotation содержит информацию о метаразметке текста и содержит следующие поля:

— meta_id – уникальный идентификатор записи метаразметки, PK (первичный ключ, англ. primary key), тип данных integer (целое);

— title – название текста, string (строка);

— author – информация об авторе текста, string (строка);

— source – информация об источнике текста, string (строка);

— creation_year – год создания текста, integer (целое);

— add_date – дата добавления текста в базу данных, date (дата).

Информация о жанрах хранится в таблице genre. Она имеет следующие поля: genre_id – уникальный идентификатор языка, PK, тип данных integer (целое), name – имя жанра (например, «drama» или «thriller»), string (строка).

В таблице language хранится информация о языках, представленных в системе. В таблице имеются следующие поля:

— language_id – уникальный идентификатор языка, PK, тип данных integer (целое);
— short_name – сокращенное название языка (например, «ru» или «fr»), string (строка);
— full_name – полное название языка на английском языке (например, «Russian» или «French»), string (строка).

Информация о языковых парах хранится в таблице language_pair, которая имеет следующие поля:

— language_pair_id – уникальный идентификатор языка, PK, тип данных integer (целое);
— from_language – исходный язык (с которого производится перевод), FK (внешний ключ, англ. foreign key) на поле language_id таблицы language, integer (целое);
— to_language – целевой язык (на который производится перевод), FK на поле language_id таблицы language, integer (целое).

Хранящиеся в базе тексты представлены в таблице text со следующими полями:

— text_id – уникальный идентификатор текста, PK, тип данных integer (целое);
— meta_annotation – информация о метаразметке, FK на поле meta_id таблицы meta_annotation, integer (целое);
— language_pair – информация о направлении перевода (исходном и целевом языках текстов), FK на поле language_pair_id таблицы language_pair, integer (целое).
— added_by – указание на пользователя, который добавил текст, FK на поле user_id таблицы user, uuid.

Поскольку метаразметка и жанры связаны связью «многие-ко-многим»,

необходимо формализовать её с помощью дополнительной таблицы meta_genre, которая имеет следующие поля:

— mg_id – уникальный идентификатор отношения текста и жанра, PK, тип данных integer (целое);
— meta_id – идентификатор метаразметки, FK на поле meta_id таблицы meta_annotation, тип данных integer (целое);
— genre_id – идентификатор жанра, FK на поле genre_id таблицы genre, тип данных integer (целое);

Информация о выровненных предложениях хранится в таблице sentence, имеющей следующие поля:

— sentence_id – уникальный идентификатор предложения, PK, тип данных integer (целое);
— source_text – предложение на исходном языке, string (строка);
— aligned_translation – перевод предложения, string (строка);
— source_text_id – идентификатор текста, из которого взято предложение, FK на поле text_id таблицы text, integer (целое).

Вся морфологическая разметка слов содержится в таблице morphological_annotation, которая имеет следующие поля:

— morph_id – уникальный идентификатор морфологической аннотации, PK, тип данных integer (целое);
— part_of_speech - часть речи, string (строка);
— grammatical_category – грамматическая категория, string (строка);
— anim – одушевленность слова, string (строка).

В таблице word хранится информация о выровненных парах слов: слову на изначальном языке приписывается его перевод и морфологические признаки слова (в исходном языке). Таблица word содержит следующие поля:

— word_id – уникальный идентификатор слова, PK, тип данных integer (целое);
— source_word – слово на исходном языке, string (строка);

— aligned_word – выровненный перевод слова, string (строка);
— sentence – идентификатор предложения, из которого взято слова (его контекст), FK на поле sentence_id таблицы sentence, integer (целое);
— morphological_annotation – идентификатор морфологической разметки слова, FK на поле morpho_id таблицы morphological_annotation, integer (целое).

Что касается пользователей, необходимо сначала рассмотреть способ хранения стран в базе данных. Для этого используется таблица country с полями country_id – уникальный идентификатор страны, PK, тип данных integer (целое) и name – название страны, string (строка).

Наконец, информация о пользователях базы хранится в таблице user со следующими полями:

— user_id – уникальный идентификатор пользователя, PK, тип данных uuid;
— name – имя пользователя, строка (string);
— surname – фамилия пользователя, строка (string);
— email – электронный адрес пользователя, строка (string);
— country – страна пользователя, FK на поле country_id таблицы country, integer (целое);
— native_language – родной язык пользователя, FK на поле language_id таблицы language, integer (целое).

2.2 Ограничения целостности базы данных

Для поддержания целостности базы данных введены ограничения на некоторые атрибуты. Разработанные ограничения представлены в таблице 2. Первичный ключ обозначен как PK (данные в столбце уникальны и не null), внешний ключ как FK (должна существовать запись в другой таблице с внешним ключом в качестве значения указанного атрибута).

Таблица 2 – Ограничения целостности базы данных

Таблица	Атрибут	Ограничение
text	text_id	PK
	meta_annotation	FK
	language_pair	FK
	added_by	FK
meta_genre	mg_id	PK
	meta_id	FK
	genre_id	FK
genre	genre_id	PK
sentence	sentence_id	PK
	text_id	FK
word	word_id	PK
	sentence_id	FK
	morpholo-gical_annotation	FK
morphological_annotation	morph_id	PK
language	language_id	PK
language_pair	language_pair_id	PK
	from_language	FK
	to_language	FK
country	country_id	PK
user	user_id	PK
	country	FK
	native_language	FK
meta_annotation	meta_id	PK
	creation_year	От 1700 до текущ. года
	add_date	От мая 2023 до текущ. даты

2.3 Разработка алгоритмов

Основная цель базы данных – поиск слов с их переводом и примерами употребления. Схема базового алгоритма поиска слова в корпусе (без фильтров) представлена на рисунке 10.

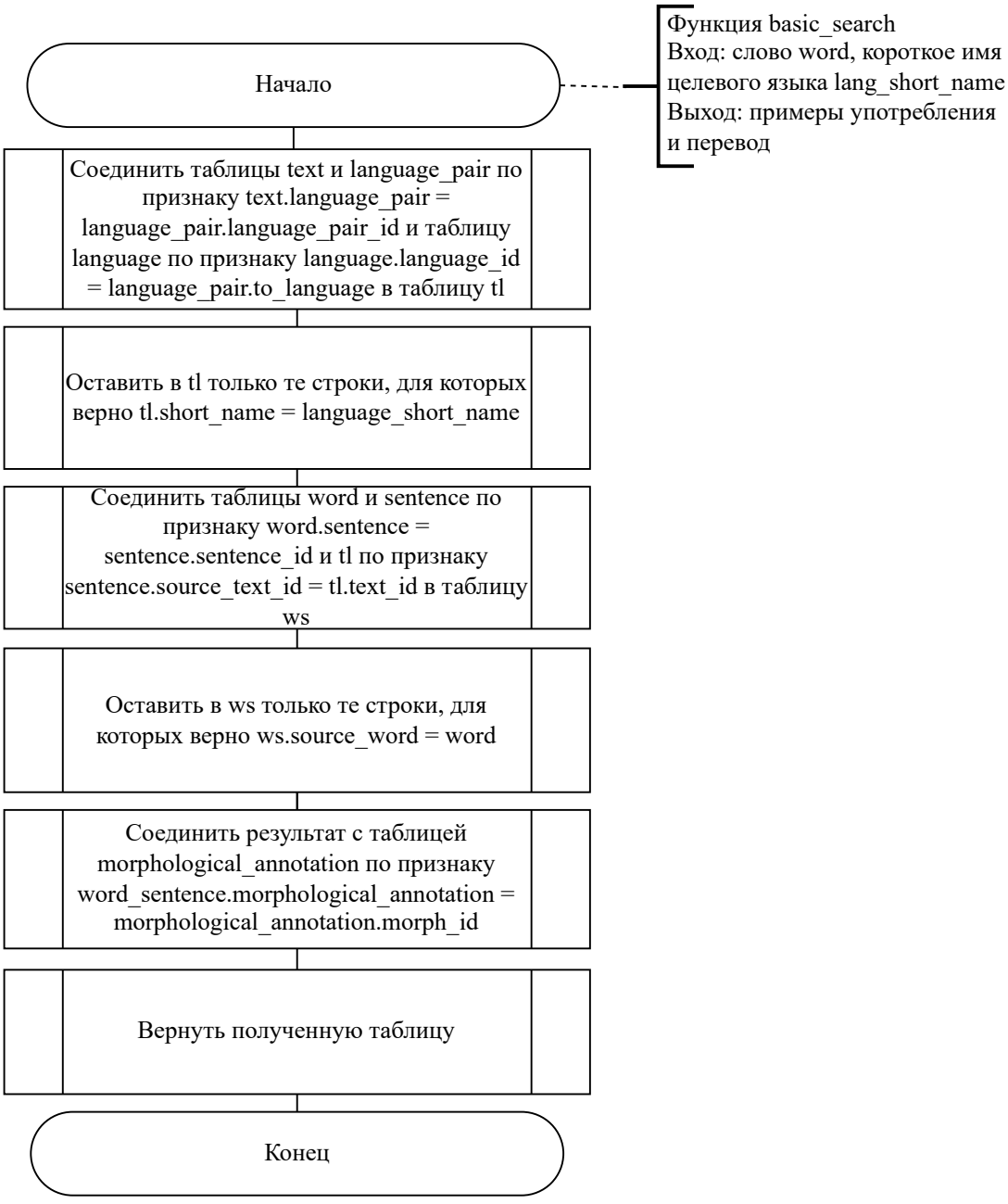

Рисунок 10 – Схема базового алгоритма поиска

Схема алгоритма поиска слова в корпусе среди текстов указанного автора, представлена на рисунке 11.

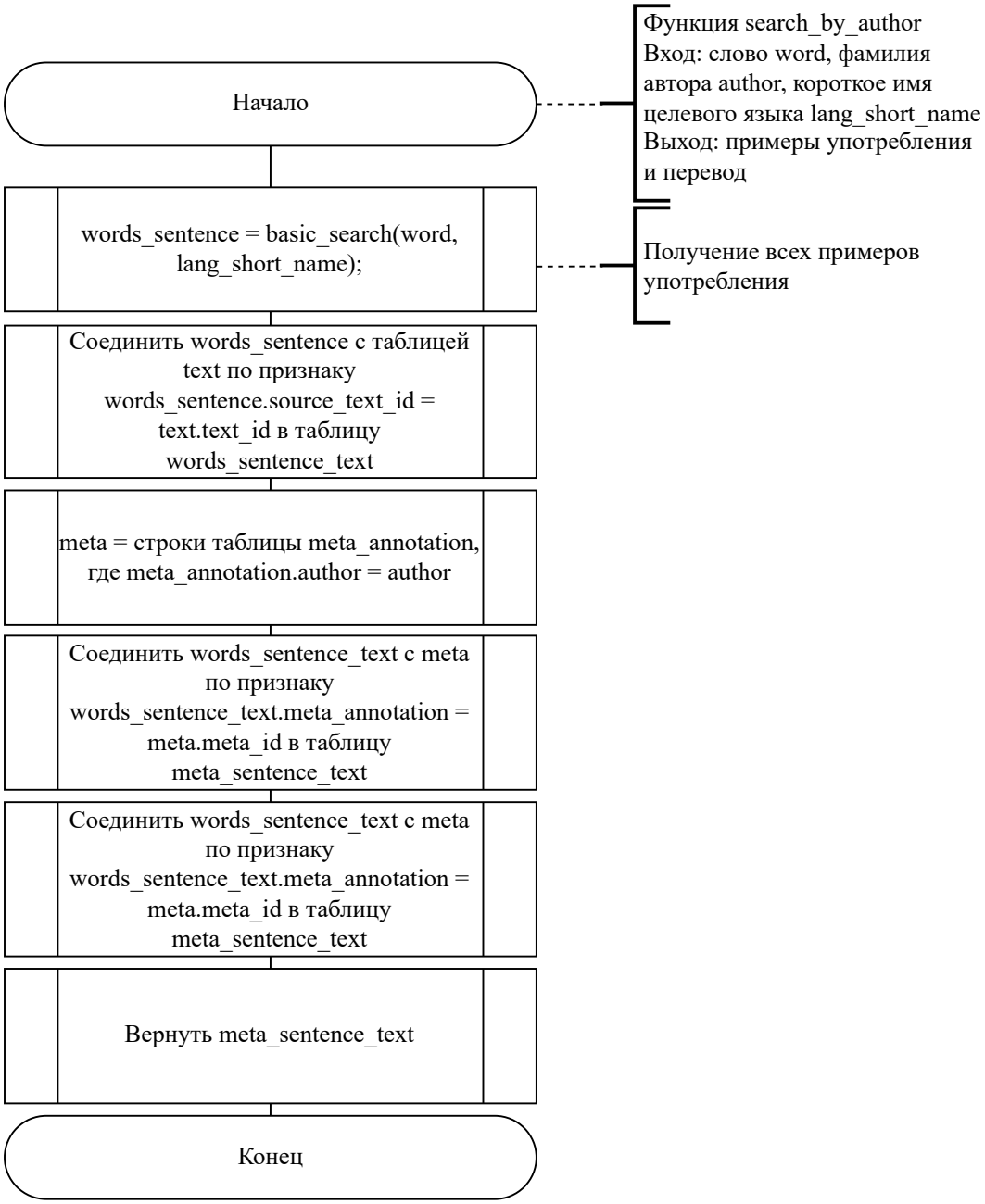

Рисунок 11 – Схема алгоритма поиска слова в текстах автора

Схема алгоритма поиска примеров употребления слова в корпусе по дате добавления текста представлена на рисунке 12.

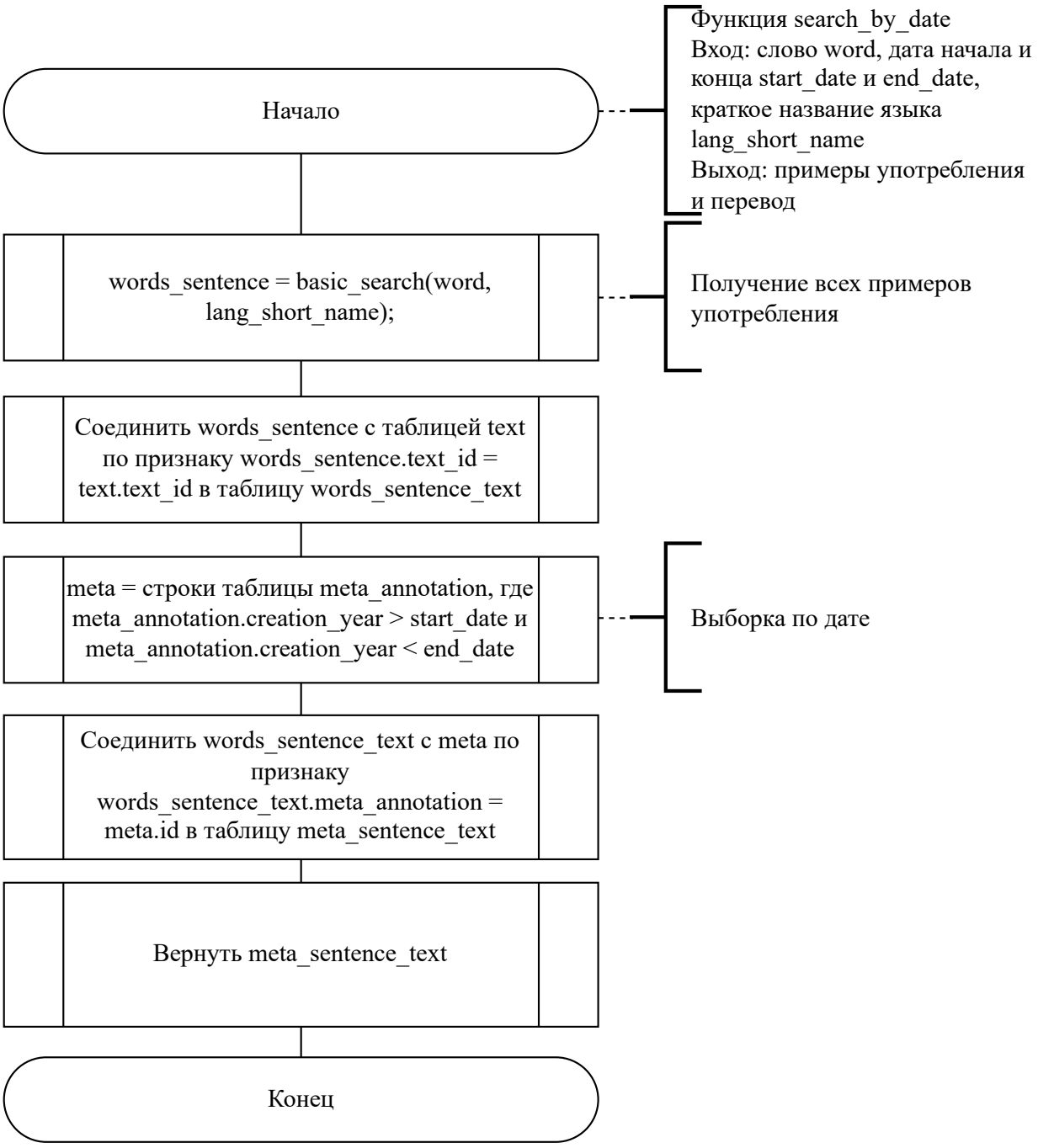

Рисунок 12 – Схема алгоритма поиска слова в текстах по дате

Схема алгоритма поиска с учетом заданной части речи представлена на рисунке 13.

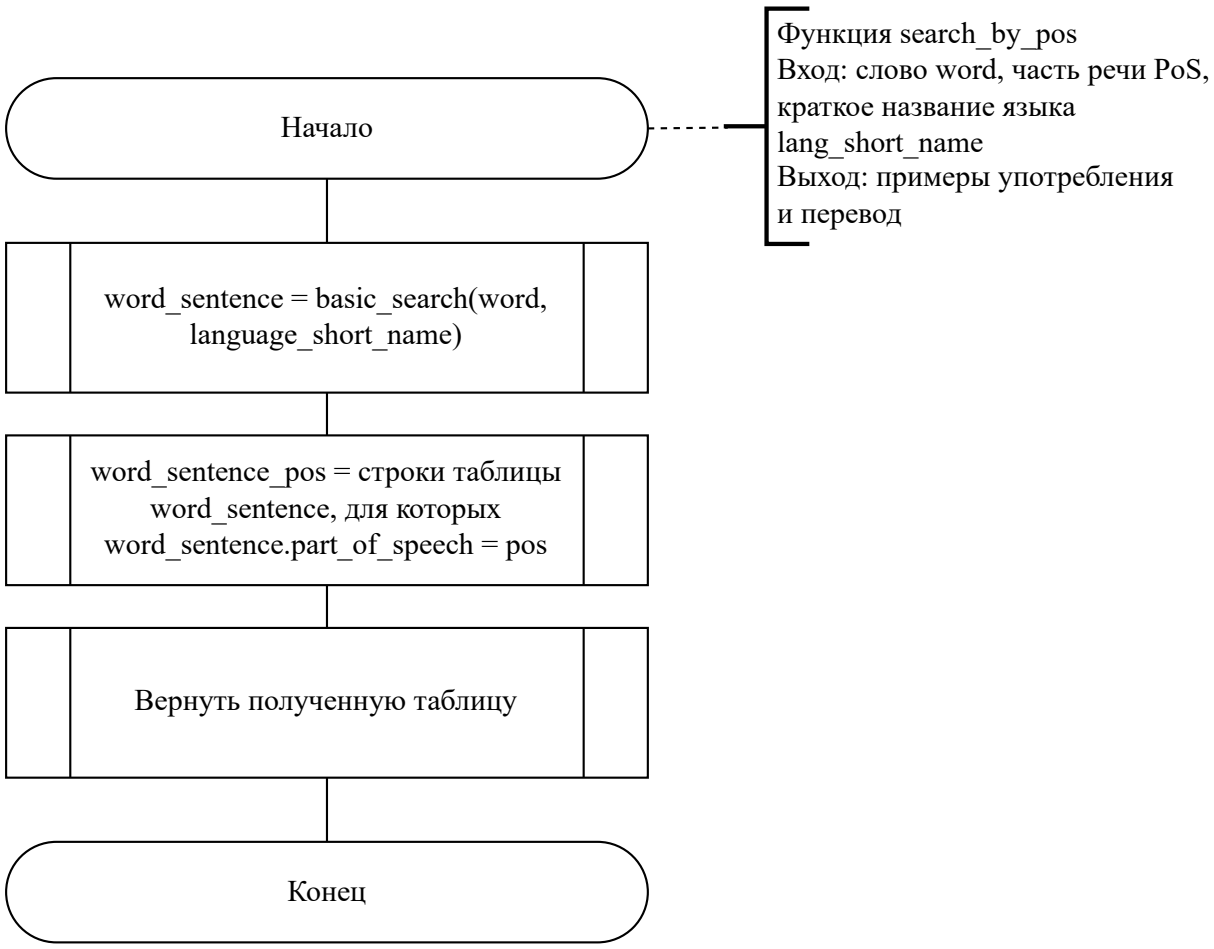

Рисунок 13 – Схема алгоритма поиска слова с учетом заданной части речи

Алгоритмы поиска по другим морфологическим признакам строятся аналогично (функции search_by_anim, search_by_gram).

Схема алгоритма поиска слова в корпусе среди текстов заданного жанра представлена на рисунке 14.

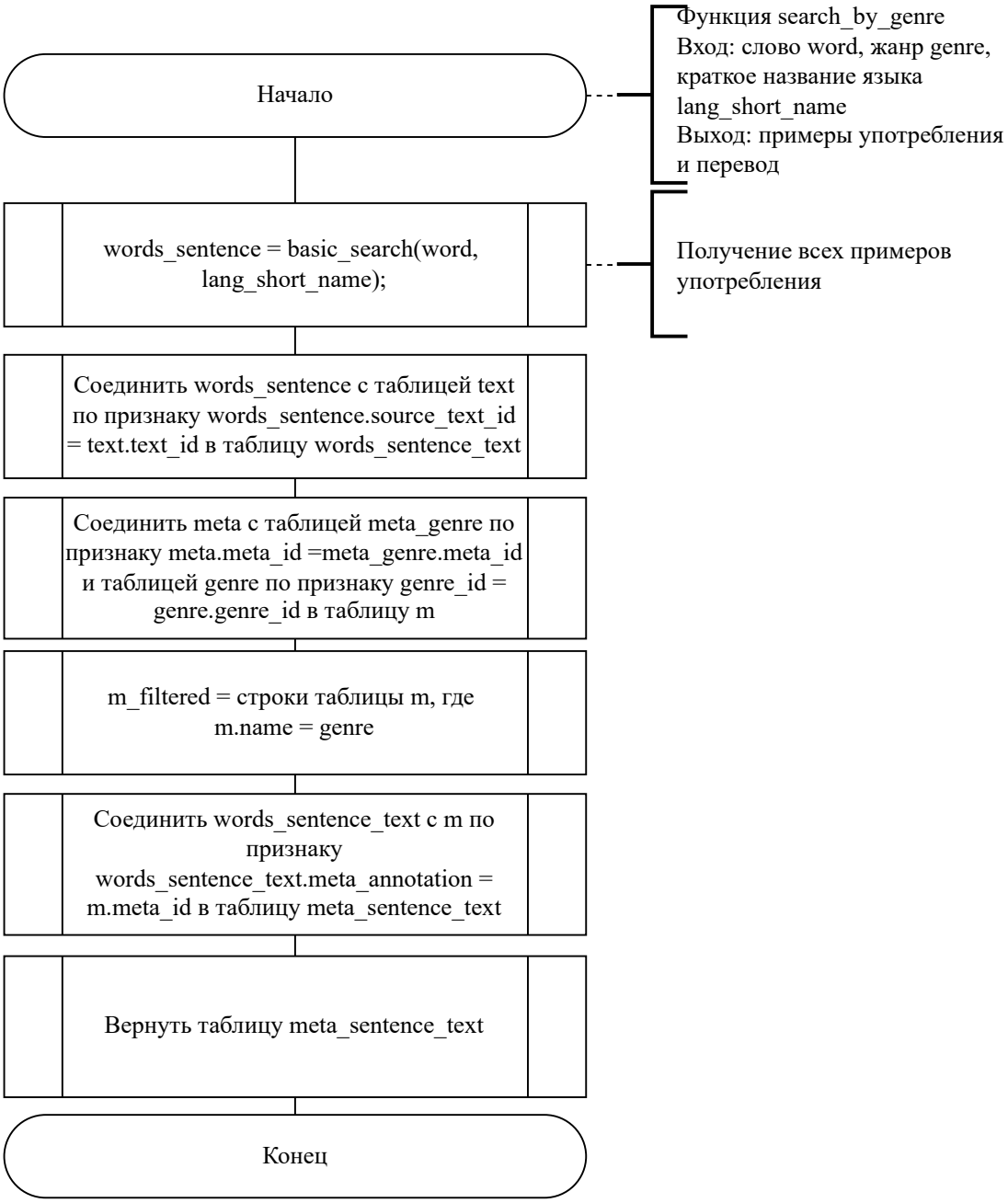

Рисунок 14 – Схема алгоритма поиска слова с учетом заданного жанра текстов

2.4 Проектирование кеширования запросов

Как видно из схем спроектированных алгоритмов, для каждого запроса необходимо выполнять определенное количество соединений таблиц, что является трудоемкой задачей. Поэтому, для исключения повторных вычислений и, соответственно, ускорения поиска, рационально использовать дополнительную кеширующую базу данных. В этом хранилище необходимо сохранять результаты запросов и собственно сами запросы, чтобы можно было потом найти ранее вычисленный результат. Запрос можно сохранять в виде структуры: слово и фильтры. Слово – строка. Фильтры задаются набором полей, которые могут быть null (поскольку не в каждом запросе заданы все параметры). Результат также задается в формате структуры (словарь или таблица – в зависимости от формата хранения данных). Кеш будет становиться недействительным по истечении 24 часов. Таким образом, ключи будут иметь следующий формат: {слово}-{язык}-{жанр}-{время начала}-{время конца}-{часть речи}-{грамм. категория}-{одушевлен-ность}-{автор}.

2.5 Проектирование компонентов системы

Диаграмма компонентов проектируемой системы в нотации UML представлена на рисунке 15.

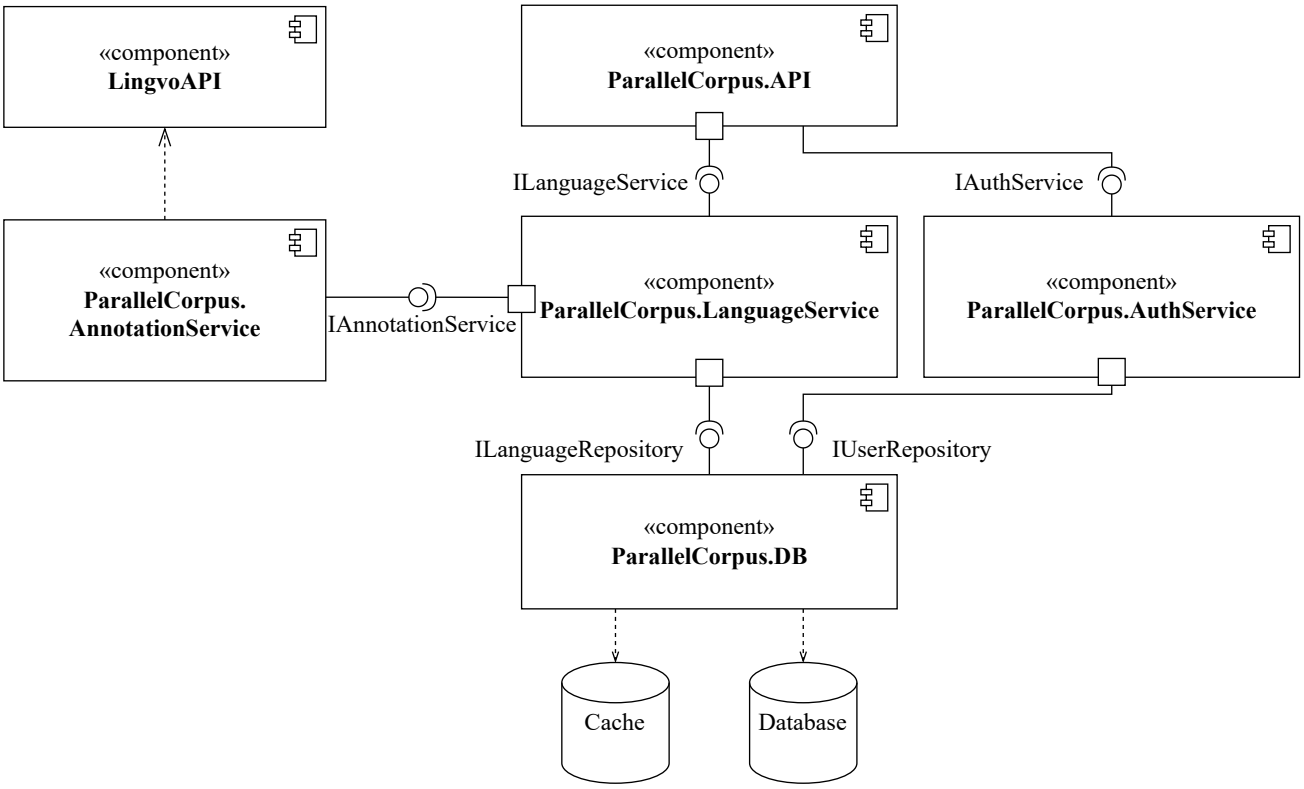

Рисунок 15 – Диаграмма компонентов системы

Система разрабатывается для хранения и пополнения параллельного корпуса текстов на естественных языках. Таким образом, выделяются следующие компоненты: ParallelCorpus.LanguageService, реализующий все возможные действия с корпусом через интерфейс ILanguageService, ParallelCorpus.AnnotationService, отвечающий за разметку текстов. Взаимодействие других компонентов с ParallelCorpus.AnnotationService осуществляется через интерфейс IAnnotationService. При этом, этот компонент является оберткой для функций компонента LingvoAPI обработки лингвистических данных.

Поскольку в системе также разграничиваются типы пользователей и присутствует функционал регистрации, выделяется сервис авторизации ParallelCorpus.AuthService, взаимодействие с которым осуществляется с помощью интерфейса IAuthService.

Все действия приложения с данными реализованы в компоненте ParallelCorpus.DB. Он предоставляет интерфейс для взаимодействия с помощью интерфейсов ILanguageRepository и IUserRepository, отвечающих за данные кор-

пуса и пользователей соответственно.

За взаимодействие пользователей (клиентов) с системой отвечает компонент ParallelCorpus.API.

2.6 Протоколирование

Протоколирование (логирование) событий, происходящих в системе, позволяет разработчикам, в случае возникновения ошибок, воспроизводить и оперативно их устранять. Для того, чтобы это было возможным, необходимо обеспечить достаточную детализацию происходящих действий. Это достигается протоколированием всех действий пользователя, в том числе запросов к базе данных с указанием используемых данных. Однако необходимо принять во внимание, что такое подробное логирование может привести, во-первых, к увеличению потребляемой памяти, и, во-вторых, к протоколированию чувствительных данных [42]. Первая проблема решается конфигурированием уровня логирования, вторая – внимательным отслеживанием данных, передающихся в логи.

Для данной системы выбраны следующие события, которые необходимо логировать:

— все внешние запросы к системе с указанием id пользователя, запрашиваемого сервиса и передаваемых данных;
— добавление, удаление, извлечение и изменение данных в базе данных;
— все вызовы сторонних сервисов (например, сервиса обработки лингвистических данных) с указанием вызываемого функционала и передаваемых данных;
— ответы на запросы к базе данных и компонентам системы;
— все ошибки, исключения и коды возврата.

2.7 Проектирование ролевой модели

Ранее введены следующие роли: неавторизованный пользователь, преподаватель, переводчик и администратор.

Неавторизованный пользователь не имеет доступа ни к каким таблицам, кроме таблицы user, в которую он может добавлять данные для обеспечения возможности регистрации и авторизации.

Администратор, напротив, имеет доступ ко всем таблицам на их чтение, а также на изменение, удаление и добавление в них данных.

Переводчик имеет права на чтение, добавление, изменение и удаление данных из таблиц text, genre, text_genre, sentence, word, morphological_annotation, meta_annotation. Причем изменение и удаление возможно только для тех данных, которые ранее были добавлены данным пользователем. Доступ переводчика к таблицам language, country и language_pair возможен только для чтения.

Роль преподавателя отличается от переводчика отсутствием прав на добавление, изменение и удаление информации в таблицах text, genre, text_genre, sentence, word, morphological_annotation, meta_annotation – однако возможность чтения информации сохраняется.

Никто, кроме администратора, не имеет права изменять структуру таблиц, удалять их или добавлять новые таблицы.

Вывод

В данном разделе представлена диаграмма проектируемой базы данных. Для формализации отношения «многие-ко-многим» выделена дополнительная сущность. Для всех таблиц описаны их атрибуты. Кроме того, описаны ограничения целостности базы данных: в каждой таблице выделен первичный ключ,

описаны внешние ключи, а также описана необходимая валидация для некоторых полей.

Разработан алгоритм функции поиска как без фильтров, так и по каждому фильтру. Также в виде схемы представлен обобщенный алгоритм поиска, объединяющий все функции.

Описан требуемый формат кеширования данных: необходимо сохранение как запроса, так и результата. Кеширование позволит избежать повторных соединений таблиц для одного и того же запроса. Также описаны события в системе, которые необходимо записывать в журнал: это все запросы пользователя, ответы на них, запросы к базе данных и все ошибки в системе.

Представлена диаграмма компонентов приложения к базе данных, в которую включены компоненты обработки запросов, выравнивания текстов и морфологической разметки слов.

Наконец, спроектирована ролевая модель на уровне базы данных. Для каждой из выделенных ролей: неавторизованный пользователь, переводчик, преподаватель и администратор, описаны права доступа к таблицам базы данных.

3 Технологический раздел

3.1 Средства реализации системы

3.1.1 Выбор СУБД

В качестве наиболее подходящей системы управления базами данных выбрана система PostgreSQL [27] – объектно-реляционная СУБД с открытым кодом. Выбор основан на исследованиях [28], которые утверждают, что самыми быстрыми по операции чтения реляционными СУБД являются MySQL [29], PostgreSQL и SQL Server [30]. При этом SQL Server, в отличие от MySQL и PostgreSQL является коммерческой СУБД, что является препятствием для её использования в данном проекте. Что касается MySQL, то в данной СУБД отсутствует возможность создания табличных функций [31], что не позволит реализовать спроектированные алгоритмы.

Что касается кеширующей базы данных, выбор пал на In-Memory Database (IMDB) Redis [37]. Согласно тем же исследованиям [28], она является одной и самых быстрых NoSQL СУБД, и, поскольку не предъявляются требования к наличию хранимых процедур для кеширующей базы данных, Redis предоставляет весь необходимый функционал.

3.1.2 Выбор языка программирования

Во многих современных языках программирования реализованы библиотеки для работы с PostgreSQL. Не исключением является и язык C# [33], выбранный в качестве основного языка программирования в данном проекте. Данный выбор связан с тем, что C# – один из наиболее активно развивающихся [32]

кросс-платформенных языков программирования с открытым исходным кодом. Множество фреймворков и библиотек на платформе .NET позволяют решить практически любую задачу.

Однако для решения поставленной задачи требуются средства обработки естественного языка. Язык C# не обладает достаточным набором средств NLP. Таким образом, часть функций необходимо реализовать на другом языке программирования. Для данных целей наиболее удачно подходит язык Python [34], библиотеки для которого предоставляют огромные возможности для анализа данных – в том числе, и лингвистических.

3.1.3 Выбор средств разработки

В качестве основной среды разработки выбрана среда JetBrains Rider 2022.3.2 [35], поскольку она предоставляет обширный набор возможностей по написанию, тестированию и отладке приложений на языке C#, а также встроенные статические и динамические анализаторы и отдельное окно для подключения к базам данных. Дополнительным преимуществом является то, что данная среда является бесплатной для студентов.

Для разработки компонента обработки лингвистических данных была выбрана среда JetBrains PyCharm 2023.1 [36]. Как и Rider, PyCharm является бесплатной для студентов и предоставляет полный набор инструментов для разработки приложений на языке Python.

3.2 Реализация базы данных

Для инициализации базы данных написаны SQL-сценарии, выполняющиеся при старте СУБД. Сценарий создания таблиц представлен в листинге **??**.

Ограничения на значения столбцов таблиц представлены в листинге **??**.

Разработанная хранимая функция basic_search представлена в листинге **??**. Остальные разработанные функции можно найти в приложении А.

Таблицы country и language изначально заполнятся данными из файлов Countries.csv и Languages.csv. Сценарий заполнения этих таблиц представлен в листинге **??**.

Реализованная ролевая модель представлена в листинге **??**.

3.3 Реализация интерфейса для взаимодействия с базой данных

Интерфейс доступа к базе данных реализован в виде Web API – набора конечных точек соединения на базе фреймворка ASP.NET Core [38]. Для удобства взаимодействия с сервером написана Swagger [39] документация на все контроллеры и модели системы. На основе написанной документации доступен Swagger UI, позволяющий интерактивно взаимодействовать с API. Начальная страница Swagger UI разработанного приложения представлена на рисунке 16.

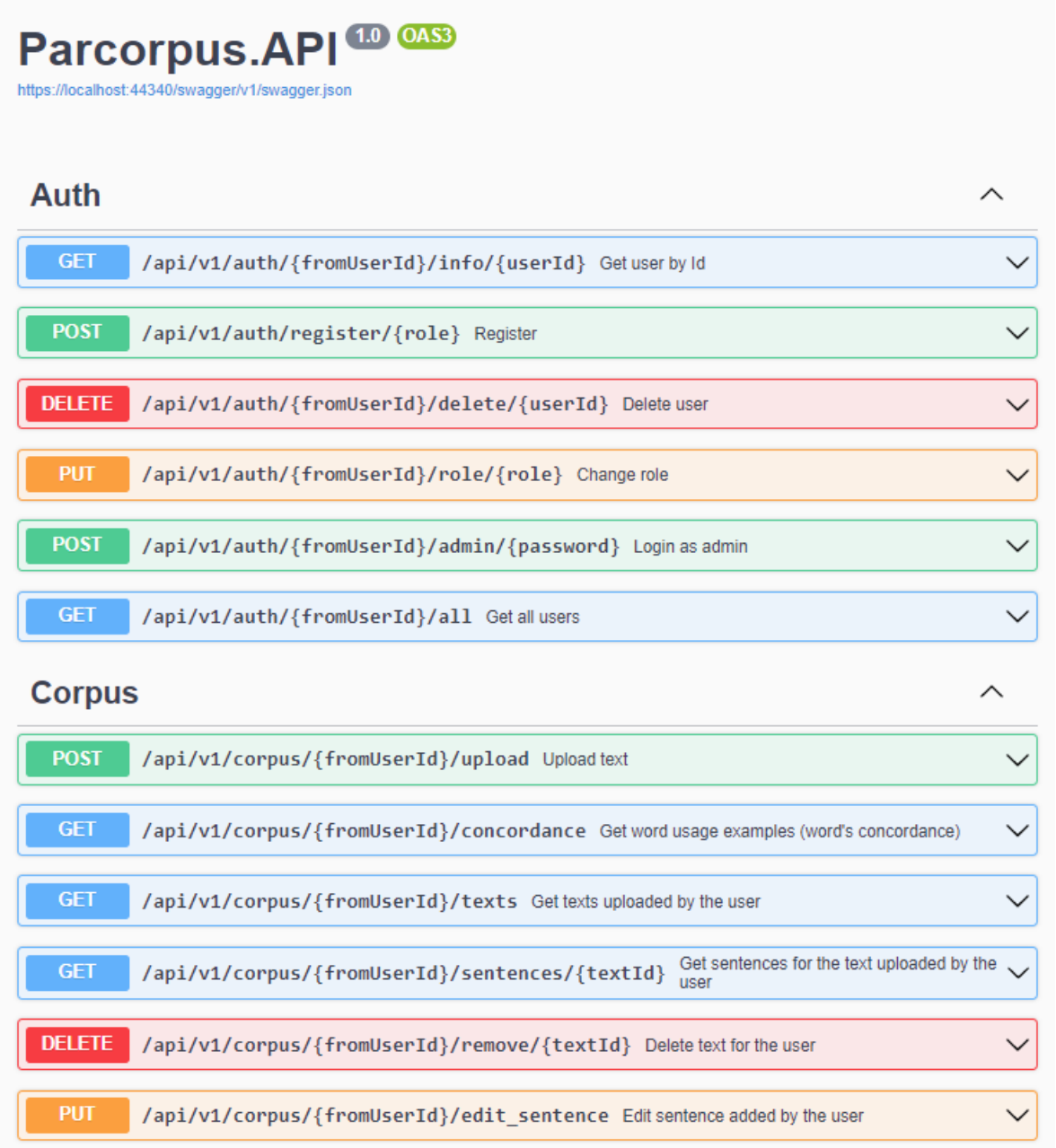

Рисунок 16 – Swagger UI

Чтобы сделать какое-либо действие, необходимо выбрать секцию с необходимой конечной точкой и ввести необходимые данные. Пример регистрации пользователя представлен на рисунке 17.

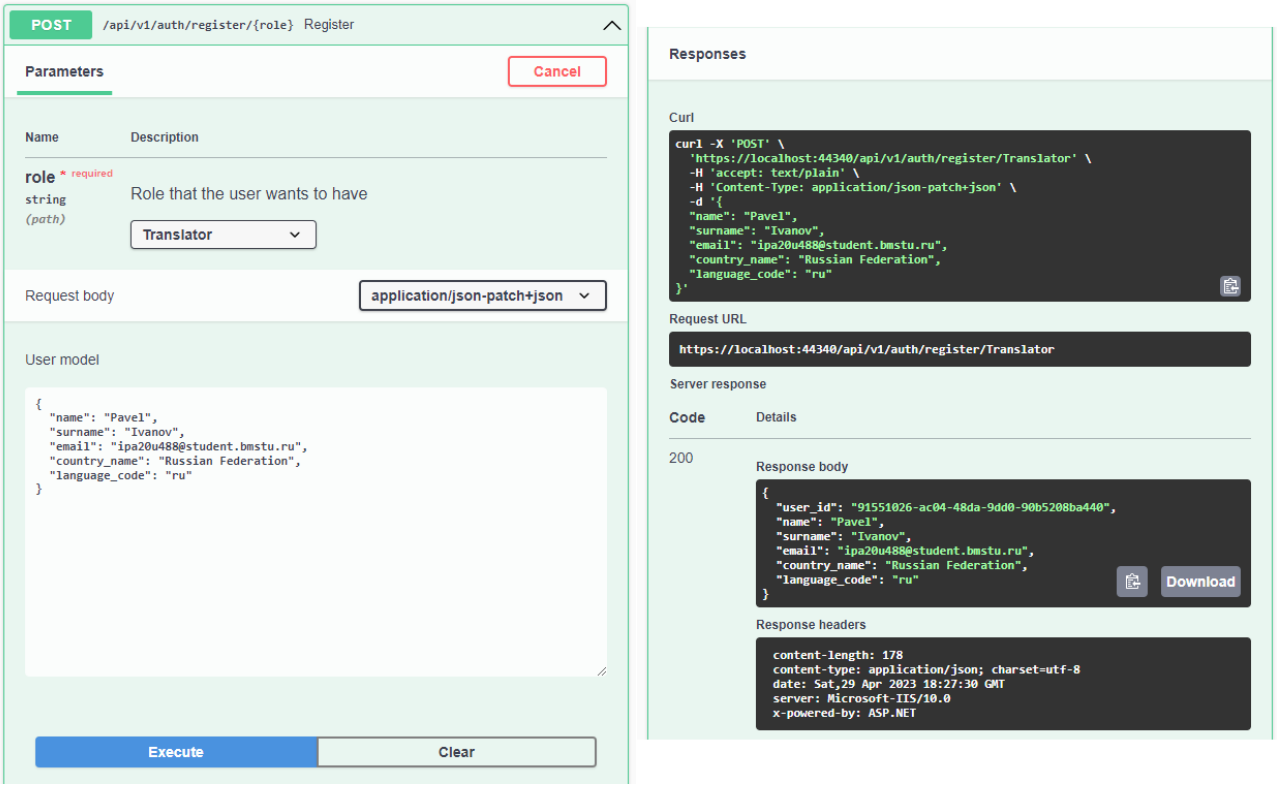

Рисунок 17 – Пример регистрации пользователя

Пример получения примеров употребления слова «и» на английском языке представлен на рисунке 18. При данном запросе прочие фильтры отсутствуют, поэтому некоторым поля результата присвоено значение null.

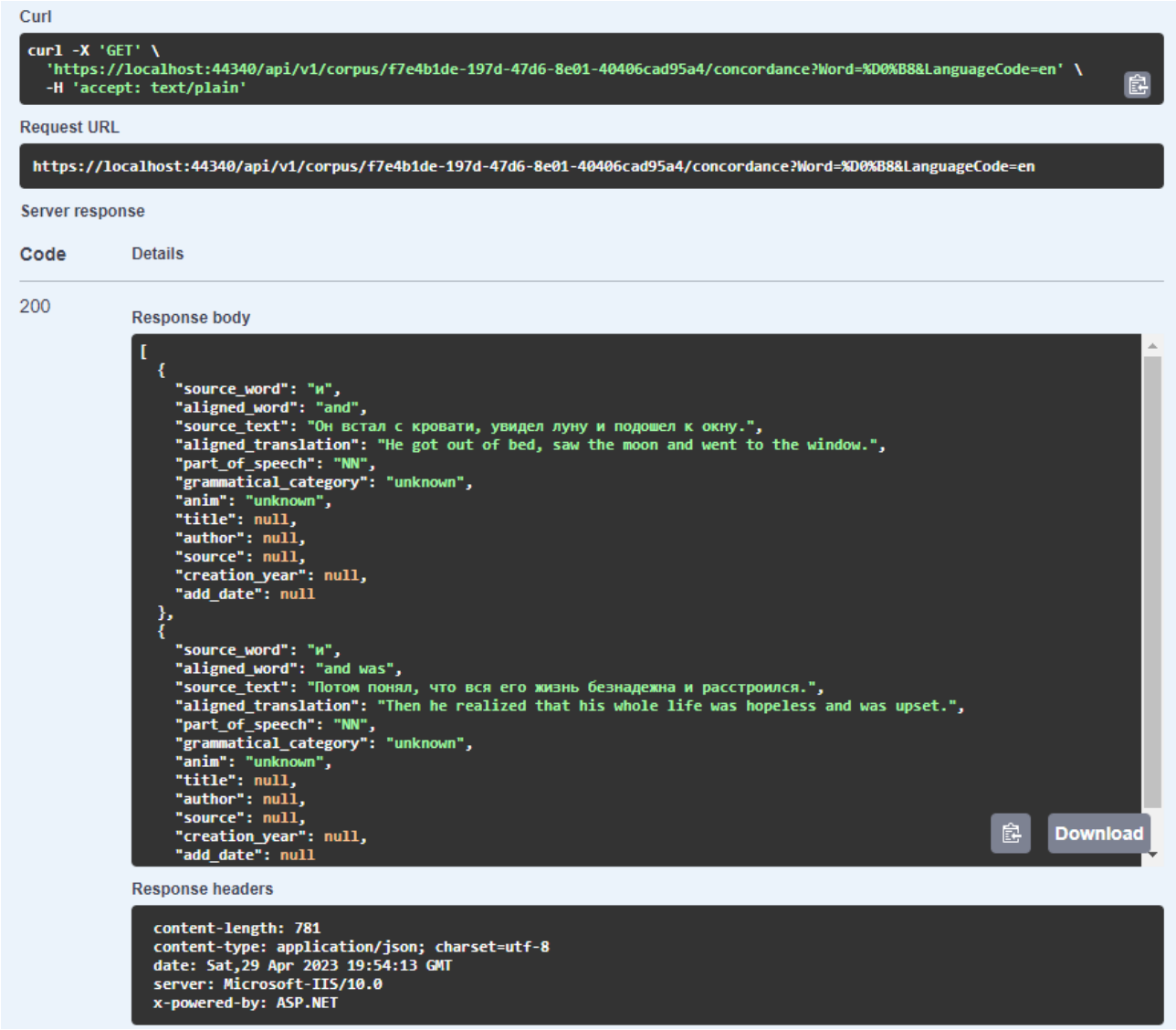

Рисунок 18 – Пример получения примеров употребления слова

3.4 Логирование действий пользователей

Для логирования действий пользователей используется платформа NLog – бесплатный фреймворк для журналирования, совместимый со стеком ELK (Elasticsearch, Logstash, Kibana) [40]. Таким образом, генерируемые логи можно использовать в поисковых и аналитических движках. Фрагмент файла конфигурации представлен в листинге 1.

Листинг 1 – Фрагмент файла NLog.config

```
<target name="asyncAllfile"
        xsi:type="AsyncWrapper"
        timeToSleepBetweenBatches="0"
        overflowAction="Discard"
        batchSize="300" >
    <!-- write logs to file  -->
    <target name="allfile"
            xsi:type="File"
            encoding="utf-8"
            fileName="\${basedir}/log/\${shortdate}.log"
            layout="\${longdate}|\${var:logId}\${guid}|
                    \${level:uppercase=true}|\${var:machineName}|
                    \${message}|
                    \${exception:maxInnerExceptionLevel=7}|
                    \${callsite}"
            archiveFileName="\${basedir}/log/archives/{#}.log"
            archiveEvery="Day"
            archiveNumbering="Date"
            archiveDateFormat="yyyy-MM-dd"
            concurrentWrites="true"/>
</target>
```

Логи сохраняются в txt файлах в папке log с именем, соответствующем текущей дате в формате «yyy-MM-dd». Автоматическое архивирование логов происходит каждый день. Формат записи сообщения в журнале предусматривает запись о дате, уникальном идентификаторе лога, уровне логирования, названии сервера, исключении (с максимальным уровнем вложенности 7) и компоненте, в котором происходит журналирование.

Пример сообщений в журнале при регистрации пользователя представлен в листинге 2.

Листинг 2 – Фрагмент логов при регистрации пользователя

```
2023-04-30 11:37:40.7594|LogId:cf0cc6a99e144680a79cf6727e9c7054|
INFO|Server|Request starting HTTP/2 POST
https://localhost:44340/api/v1/auth/register/Translator
   application/json-patch+json 150||
Microsoft.AspNetCore.Hosting.HostingApplicationDiagnostics.LogRe-
   questStarting
...
2023-04-30 11:37:44.5283|LogId:0417512af22c409586a443f95579b39e|
INFO|Server|Executed DbCommand (72ms)
   [Parameters=[@__countryName_0='?'], CommandType='Text',
   CommandTimeout='30']
SELECT c.country_id, c.name
FROM parcorpusdb.country AS c
WHERE c.name = @__countryName_0
LIMIT 1||Microsoft.EntityFrameworkCore.Diagnostics
.EventDefinition`6.Log

2023-04-30 11:37:44.6832|LogId:6c5d01f84b954671ad9ac7e057cbed29|
INFO|Server|Executed DbCommand (8ms)
   [Parameters=[@__languageShortName_0='?'], CommandType='Text',
   CommandTimeout='30']
SELECT l.language_id, l.full_name, l.short_name
FROM parcorpusdb.language AS l
WHERE l.short_name = @__languageShortName_0
LIMIT 1||
Microsoft.EntityFrameworkCore.Diagnostics
.EventDefinition`6.Log

2023-04-30 11:37:45.1292|LogId:37342430a3fa4fdc9bc2fc05177f4ed9|
INFO|Server|Executed DbCommand (55ms) [Parameters=[@p0='?'
   (DbType = Guid), @p1='?' (DbType = Int32), @p2='?', @p3='?',
   @p4='?' (DbType = Int32), @p5='?'], CommandType='Text',
   CommandTimeout='30']
```

Окончание листинга 2

```
INSERT INTO parcorpusdb."user" (user_id, country, email, name,
   native_language, surname)
VALUES (@p0, @p1, @p2, @p3, @p4, @p5);||
Microsoft.EntityFrameworkCore.Diagnostics.EventDefinition`6.Log

2023-04-30 11:37:53.2174|LogId:3ce6e47b6a504f49bc8e497ec696801c|
INFO|Server|Executed DbCommand (17ms) [Parameters=[],
   CommandType='Text', CommandTimeout='30']
create user "5182a73d-194a-4665-9651-a8c862571daf";||
Microsoft.EntityFrameworkCore.Diagnostics.EventDefinition`6.Log

2023-04-30 11:37:53.2891|LogId:a0aaf2140d264e06b72d430597aee1b2|
INFO|Server|Executed DbCommand (64ms) [Parameters=[],
   CommandType='Text', CommandTimeout='30']
grant translator to "5182a73d-194a-4665-9651-a8c862571daf";||
Microsoft.EntityFrameworkCore.Diagnostics.EventDefinition`6.Log
...
2023-04-30 11:37:53.4192|LogId:221418d9b0ba4538b118b2fd5d90a091|
INFO|Server|Request finished HTTP/2 POST
   https://localhost:44340/api/v1/auth/register/Translator
   application/json-patch+json 150 - 200 178
   application/json;+charset=utf-8 12663.1748ms||
Microsoft.AspNetCore.Hosting.HostingApplicationDiagnostics
.LogRequestFinished
```

3.5 Инструкция для развертывания системы

Для развертывания разработанной системы необходимо установить Docker – программное обеспечение для автоматизации развертывания и управления приложениями в средах с поддержкой контейнеризации [41]. Docker доступен как на Windows, так и на Linux или MacOS.

Для запуска системы на компьютере или на сервере необходимо проделать следующие действия:

1) открыть терминал в директории приложения (в ней находятся папки doc, src и test);
2) ввести команду «cd ./src/Scripts && docker-compose up -d».

После нажатия Enter система Docker самостоятельно скачает и установит все необходимые пакеты, а также запустит СУБД и сервисы приложения. Пример вывода консоли при запуске системы представлен в листинге 3.

Листинг 3 – Запуск системы

```
C:\Users\Pavel\Desktop\parcorpus\src\Scripts>
docker-compose up -d
[+] Running 5/5
Network scripts_backend-network Created 0.1s
Container redis                 Started 3.0s
Container aligner               Started 3.2s
Container postgres              Started 3.4s
Container parcorpus-backend     Started
```

После запуска системы её записи журнала доступны на локальной машине и находятся в папке /src/Parcorpus/Parcorpus.API/logs. Изменение настроек (например, строки подключения к базе данных или кеширующей СУБД) доступно в файле /src/Parcorpus/Parcorpus.API/appsettings.json. При изменении данного файла будет также изменен файл в контейнере.

После запуска системы взаимодействие может с ней осуществляется по адресу http://localhost:802/. Список доступных конечных точек соединения по данному адресу представлен в таблице 3.

Таблица 3 – Конечные точки соединения

Конечная точка	Параметры	Описание
GET: /api/v1/auth/ {fromUserId}/info/ {userId}	fromUserId – идентификатор пользователя, который совершает запрос; userId – идентификатор пользователя, о котором запрашивается информация.	Получение информации о пользователе
DELETE: /api/v1/auth/ {fromUserId}/delete/ {userId}	fromUserId – идентификатор пользователя, который совершает запрос; userId – идентификатор удаляемого пользователя.	Удаление пользователя
PUT: /api/v1/auth/ {fromUserId}/role/{role}	fromUserId – идентификатор пользователя, который совершает запрос; role – новая роль пользователя.	Смена роли
POST: /api/v1/auth/register/{role}	role – роль регистрируемого пользователя; Тело запроса (application/json-patch+json): UserRegistrationDto.	Регистрация нового пользователя

Продолжение таблицы 3

Конечная точка	Параметры	Описание
POST: /api/v1/auth/ {fromUserId}/admin/ {password}	fromUserId – идентификатор пользователя, который совершает запрос; password – пароль.	Авторизация администратора
GET: /api/v1/auth/ {fromUserId}/all	fromUserId – идентификатор пользователя, который совершает запрос.	Получение информации о всех пользователях
POST: /api/v1/corpus/ {fromUserId}/upload	fromUserId – идентификатор пользователя, который совершает запрос. Тело запроса (multipart/form-data): InputTextDto.	Загрузка нового текста
GET: /api/v1/corpus/ {fromUserId}/ concordance	fromUserId – идентификатор пользователя, который совершает запрос. Тело запроса (multipart/form-data): ConcordanceQueryDto.	Поиск примеров употребления слова
GET: /api/v1/corpus/ {fromUserId}/texts	fromUserId – идентификатор пользователя, который совершает запрос;	Получение всех текстов пользователя

Окончание таблицы 3

Конечная точка	Параметры	Описание
GET: /api/v1/corpus/ {fromUserId}/sentences/ {textId}	fromUserId – идентификатор пользователя, который совершает запрос; textId – идентификатор текста.	Получение всех предложений добавленного пользователем текста
DELETE: /api/v1/corpus/ {fromUserId}/remove/ {textId}	fromUserId – идентификатор пользователя, который совершает запрос; textId – идентификатор текста.	Удаление текста
PUT: /api/v1/corpus/ {fromUserId}/ edit_sentence	fromUserId – идентификатор пользователя, который совершает запрос; Тело запроса (application/json-patch+json): SentenceDto.	Редактирование предложения

Упомянутые объекты передачи данных (англ. «Data transfer object», «DTO») имеют структуру, представленную в листингах 4–8.

Листинг 4 – UserRegistrationDto

```
1 public class UserRegistrationDto
2 {
3     /// <summary>  Имя пользователя  </summary>
4     /// <example>Pavel</example>
5     [JsonProperty("name")]
6     public string Name { get; set; }
7
8     /// <summary> Фамилия пользователя </summary>
```

Окончание листинга 4

```
    /// <example>Ivanov</example>
    [JsonProperty("surname")]
    public string Surname { get; set; }

    /// <summary> Электронная почта пользователя </summary>
    /// <example>ipa20u488@student.bmstu.ru</example>
    [JsonProperty("email")]
    public string Email { get; set; }

    /// <summary> Название страны пользователя </summary>
    /// <example>Russian Federation</example>
    [JsonProperty("country_name")]
    public string CountryName { get; set; }

    /// <summary>
    /// Сокращенное название родного языка пользователя
    /// </summary>
    /// <example>ru</example>
    [JsonProperty("language_code")]
    public string LanguageCode { get; set; }
}
```

Листинг 5 – InputTextDto

```
public class InputTextDto
{
    /// <summary> Сокращенное название исходного языка </summary>
    /// <example>ru</example>
    public string SourceLanguageCode { get; set; }

    /// <summary> Сокращенное название целевого языка </summary>
    /// <example>en</example>
    public string TargetLanguageCode { get; set; }
```

Окончание листинга 5

```
10      /// <summary> Название текста </summary>
11      public string? Title { get; set; }
12
13      /// <summary> Автор текста </summary>
14      public string? Author { get; set; }
15
16      /// <summary> Источник текста </summary>
17      public string? Source { get; set; }
18
19      /// <summary> Год написания текста </summary>
20      public int? CreationYear { get; set; }
21
22      /// <summary> Перечисление жанров </summary>
23      public IEnumerable<string> Genres { get; set; }
24
25      /// <summary> Файл с исходным текстом </summary>
26      [FromForm(Name="SourceText")]
27      public IFormFile SourceText { get; set; }
28
29      /// <summary> Файл с переведенным текстом </summary>
30      [FromForm(Name="TargetText")]
31      public IFormFile TargetText { get; set; }
32  }
```

Листинг 6 – ConcordanceQueryDto

```
1  public class ConcordanceQueryDto
2  {
3      /// <summary> Слово для поиска </summary>
4      [JsonProperty("word")]
5      public string Word { get; set; }
6
7      /// <summary> Краткое название целевого языка </summary>
```

Окончание листинга 6

```
    /// <example>"ru"</example>
    [JsonProperty("language_code")]
    public string LanguageCode { get; set; }

    /// <summary> Фильтры поиска </summary>
    [JsonProperty("filter")]
    public FilterDto? Filter { get; set; }
}
```

Листинг 7 – FilterQueryDto

```
public class FilterDto
{
    /// <summary> Жанр </summary>
    [JsonProperty("genre")]
    public string? Genre { get; set; }

    /// <summary> Дата начала </summary>
    [JsonProperty("start_date_time")]
    public DateTime? StartDateTime { get; set; }

    /// <summary> Дата окончания </summary>
    [JsonProperty("end_date_time")]
    public DateTime? EndDateTime { get; set; }

    /// <summary> Часть речи </summary>
    [JsonProperty("part_of_speech")]
    public string? Pos { get; set; }

    /// <summary> Грамматическая категория </summary>
    [JsonProperty("gram")]
    public string? Gram { get; set; }

    /// <summary> Одушевленность </summary>
```

Окончание листинга 7

```
24      [JsonProperty("anim")]
25      public string? Anim { get; set; }
26
27      /// <summary> Автор </summary>
28      [JsonProperty("author")]
29      public string? Author { get; set; }
30  }
```

Листинг 8 – SentenceDto

```
 1  public class SentenceDto
 2  {
 3      /// <summary> Id предложения </summary>
 4      [JsonProperty("sentence_id")]
 5      public int? SentenceId { get; set; }
 6
 7      /// <summary>
 8      /// Новый текст предложения на исходном языке
 9      /// </summary>
10      /// <example>I saw an apple</example>
11      [JsonProperty("source_text")]
12      public string? SourceText { get; set; }
13
14      /// <summary> Новый перевод </summary>
15      /// <example>Я увидел яблоко</example>
16      [JsonProperty("aligned_translation")]
17      public string? AlignedTranslation { get; set; }
18  }
```

Общая последовательность действий при работе с системой следующая:

1) Регистрация и получение UserId.

```
curl -X 'POST' \
    'http://localhost:802/api/v1/auth/register/Translator' \
    -H 'accept: text/plain' \
    -H 'Content-Type: application/json-patch+json' \
    -d '{ \
        "name": "Peter", \
        "surname": "Parker", \
        "email": "ipa20u488@student.bmstu.ru", \
        "country_name": "Russian Federation", \
        "language_code": "ru" \
    }'
```

В качестве ответа возвращается JSON, содержащий UserId.

```
{
    "user_id": "9c95e47a-9617-4ab3-b80d-b31055a97f19",
    "name": "Peter",
    "surname": "Parker",
    "email": "ipa20u488@student.bmstu.ru",
    "country_name": "Russian Federation",
    "language_code": "ru"
}
```

2) Запомнив UserId, можно выполнять любые запросы. Например, получение примеров употребления.

```
curl -X 'GET' \
    'http://localhost:802/api/v1/corpus/ \
    9c95e47a-9617-4ab3-b80d-b31055a97f19/concordance? \
    Word=%D0%B8&LanguageCode=en&Filter.Genre=Drama' \
    -H 'accept: text/plain'
```

Полученный ответ:


```
[
    {
        "source_word": "и",
        "aligned_word": "and",
        "source_text": "Он встал с кровати, увидел луну и
                        подошел к окну.",
        "aligned_translation": "He got out of bed, saw the
                           moon and went to the window.",
        "part_of_speech": "NN",
        "grammatical_category": "unknown",
        "anim": "unknown",
        "title": "My example 1",
        "author": "Pavel Ivanov",
        "source": "Pavel Ivanov",
        "creation_year": "2023-01-01",
        "add_date": "2023-04-29"
    }, {
        "source_word": "и",
        "aligned_word": "and was",
        "source_text": "Потом понял, что вся его жизнь
                       безнадежна и расстроился.",
        "aligned_translation": "Then he realized that his
                  whole life was hopeless and was upset.",
        "part_of_speech": "NN",
        "grammatical_category": "unknown",
        "anim": "unknown",
        "title": "My example 1",
        "author": "Pavel Ivanov",
        "source": "Pavel Ivanov",
        "creation_year": "2023-01-01",
        "add_date": "2023-04-29"
    }
]
```

3) Возможна аутентификация в качестве администратора. Для этого необходимо ввести пароль: pavelivanovIU7-65B.

```
curl -X 'POST' \
    'http://localhost:802/api/v1/auth/ \
        9c95e47a-9617-4ab3-b80d-b31055a97f19/admin/ \
        pavelivanovIU7-65B' \
    -H 'accept: */*' \
    -d ''
```

В случае успеха возвращается код 200.

3.6 Тестирование

В результате тестирования ролей получены результаты, представленные в таблице 4. Как видно из полученных результатов, разработанная система соответствует спроектированной ролевой модели. В случае отсутствия прав возвращается ошибка 401. В методах sentences, remove и edit_sentence ошибка 401 возникает в случае обращения к тексту, который добавил другой пользователь (кроме случая администратора).

Таблица 4 – Результаты тестирования разделения ролей

Роль	Конечная точка	Ответ
Translator | Admin	/api/v1/corpus/{fromUserId}/upload	201 / 409
Teacher	/api/v1/corpus/{fromUserId}/upload	401
Teacher | Translator | Admin	/api/v1/corpus/{fromUserId}/concordance	200 / 404
Translator | Translator | Admin	/api/v1/corpus/{fromUserId}/texts	200 / 404
Translator | Teacher	/api/v1/corpus/{fromUserId}/sentences/{textId}	200 / 401 / 404
Admin	/api/v1/corpus/{fromUserId}/sentences/{textId}	200 / 404
Translator | Teacher	/api/v1/corpus/{fromUserId}/remove/{textId}	200 / 401 / 404
Admin	/api/v1/corpus/{fromUserId}/remove/{textId}	200 / 404
Translator | Teacher	/api/v1/corpus/{fromUserId}/edit_sentence	200 / 401 / 404
Admin	/api/v1/corpus/{fromUserId}/edit_sentence	200 / 404
Translator | Teacher	/api/v1/auth/{fromUserId}/info/{userId}	401
Admin	/api/v1/auth/{fromUserId}/info/{userId}	200 / 404

Окончание таблицы 4

Роль	Конечная точка	Ответ
Translator | Teacher	/api/v1/auth/{fromUserId}/ delete/{userId}	401
Admin	/api/v1/auth/{fromUserId}/ delete/{userId}	200 / 404
Translator | Teacher | Admin	/api/v1/auth/{fromUserId}/ admin/{password}	200 / 403
Translator | Teacher	/api/v1/auth/{fromUserId}/all	401
Admin	/api/v1/auth/{fromUserId}/all	200

Вывод

В данном разделе было приведено обоснование выбора средств реализации системы. В качестве основной реляционной СУБД была выбрана система с открытым кодом PostgreSQL, поскольку она является одной из самых быстрых (по операции чтения) СУБД. Для системы кеширования была выбрана In-Memory Database Redis, поскольку она является одной из самых быстрых NoSQL СУБД и предоставляет весь необходимый для решения поставленной задачи функционал.

Для написания интерфейса для взаимодействия с параллельным корпусом был выбран язык C#, а для обработки лингвистических данных язык Python. В результате написано REST API на основе фреймворка ASP.NET Core, предоставляющее набор конечных точек для работы с корпусом. Разработанная система реализует спроектированную ролевую модель, систему журналирования действий пользователя и использование кеша для ускорения получения ответов на выборку данных из корпуса. Тестирование разделения ролей показало, что

разработанная система соответствует спроектированной ролевой модели.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры, следующие.

— Операционная система Windows 10 [43] x86_64.

— Память 8 Гб 2400 МГц DDR4.

— 1.6 ГГц 4-ядерный процессор Intel Core i5 8265U [44].

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время проведения исследований ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой выполнения замеров.

4.2 Подготовка исследования

Для выявления зависимости времени исполнения запросов от использования кеширующей СУБД база данных была пополнена текстами на французском и немецком языках. База данных заполнена более чем тридцатью предложениями и более чем 500 словами. На момент начала замеров кеширующая СУБД пуста.

Замеры производятся для двух запросов на получение примеров употребления слова в корпусе.

1) Поиск перевода французского слова «nous» (рус. «мы») на немецкий язык: api/v1/corpus/{uuid}/concordance?Word=nous&LanguageCode=de. Далее обозначим этот запрос как «простой», поскольку он не предполагает фильтрацию.
2) Поиск перевода французского слова «nous» на немецкий язык среди тек-

стов «Описательного» жанра, написанные с 1 января 2000 по 14 мая 2023: api/v1/corpus/{uuid пользователя}/concordance?Word=nous&LanguageCode=de&Filter.Genre=Description&Filter.StartDateTime=01.01.2000&Filter.EndDateTime=05.14.2023. Обозначим этот запрос как «сложный», поскольку он потребует больше соединений таблиц.

4.3 Результаты замеров

На рисунке 19 представлено время выполнения «простого» запроса методом GetConcordance класса LanguageRepository, который выполняет проверку на наличие в кеше данных по запросу и непосредственное выполнение запросов к базе данных.

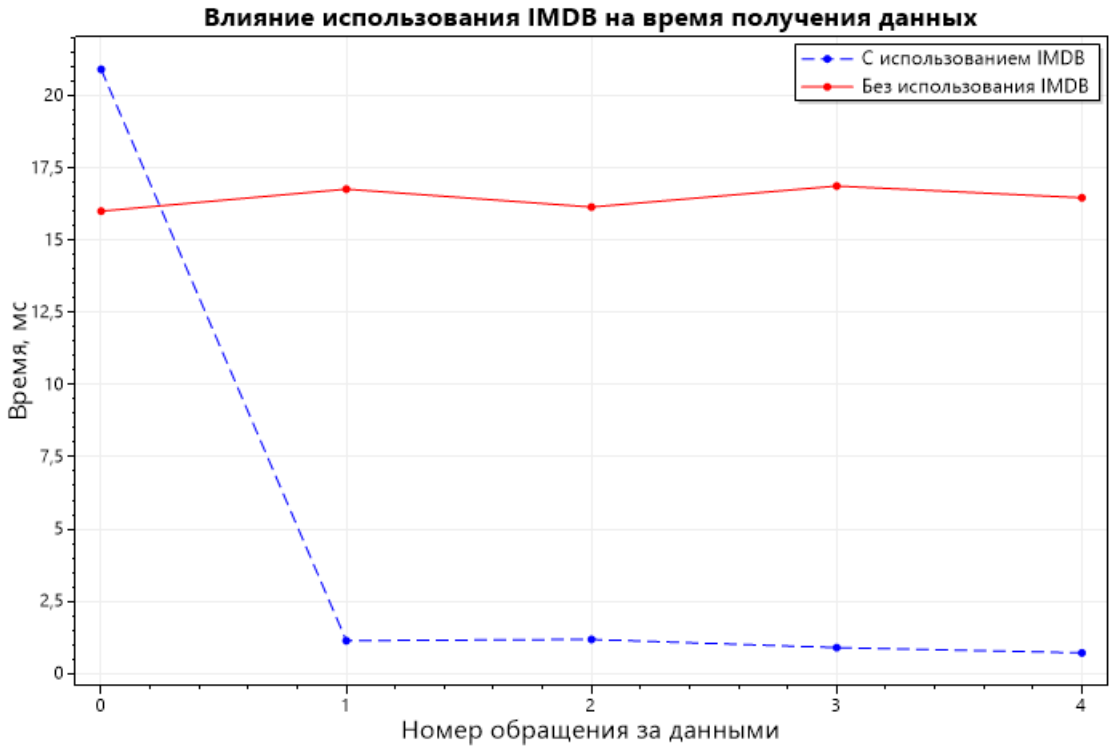

Рисунок 19 – Время выполнения «простого» запроса методом GetConcordance

Как видно из полученных данных, после первого обращения за данными время их получения уменьшается более чем в 20 раз. Время выполнения «простого» запроса без использования кеширующей СУБД не зависит от номера обращения за данными.

Аналогичные результаты получены для «сложного» запроса. Время его выполнения представлено на рисунке 20.

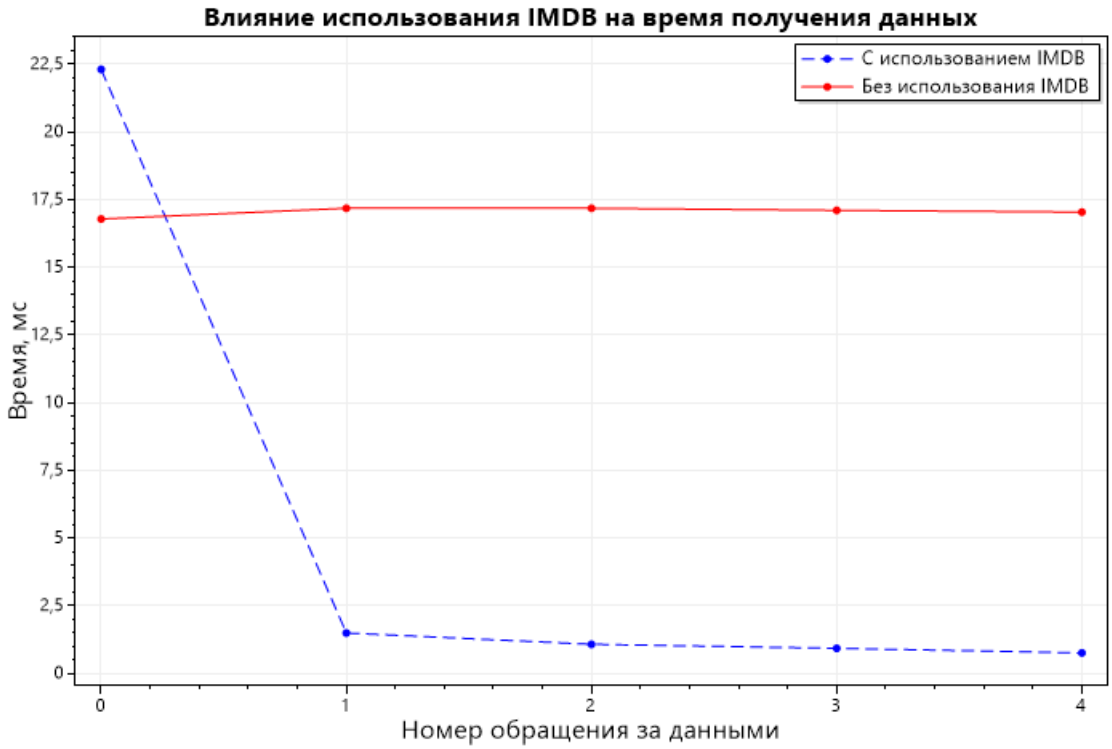

Рисунок 20 – Время выполнения «сложного» запроса методом GetConcordance

Данные, представленные на рисунках 19 и 20, получены вследствие усреднения результатов 500 замеров времени выполнения запросов с одними и теми же входными данными.

С помощью программы Apache JMeter [45] было проведено нагрузочное тестирование системы. Результаты замеров времени ответа системы на «простой» запрос с использованием и без использования кеширующей СУБД представлены на рисунках 21 и 22 соответственно.

Рисунок 21 – Время выполнения «простого» запроса к системе с использованием кеширующей СУБД

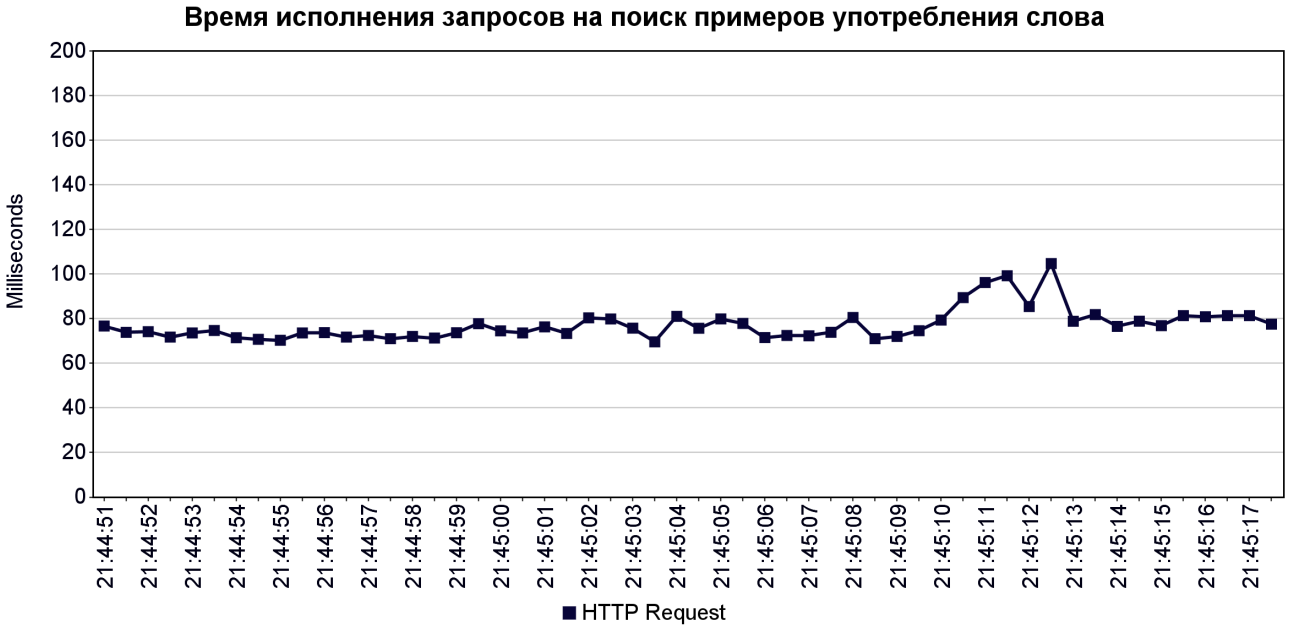

Рисунок 22 – Время выполнения «простого» запроса к системе без использования кеширующей СУБД

На рисунках 23 и 24 представлены аналогичные замеры для «сложного» запроса.

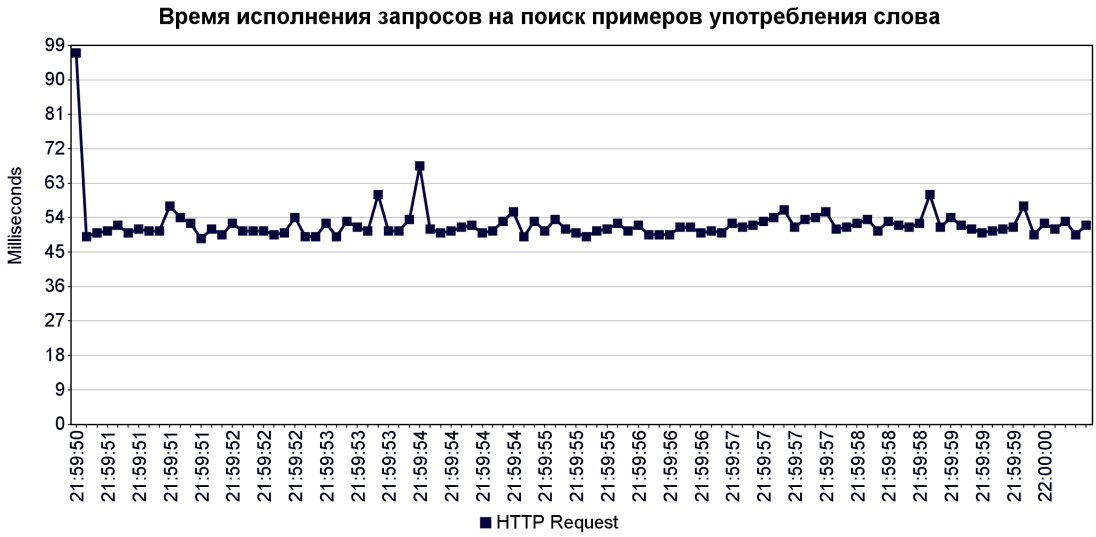

Рисунок 23 – Время выполнения «сложного» запроса к системе с использованием кеширующей СУБД

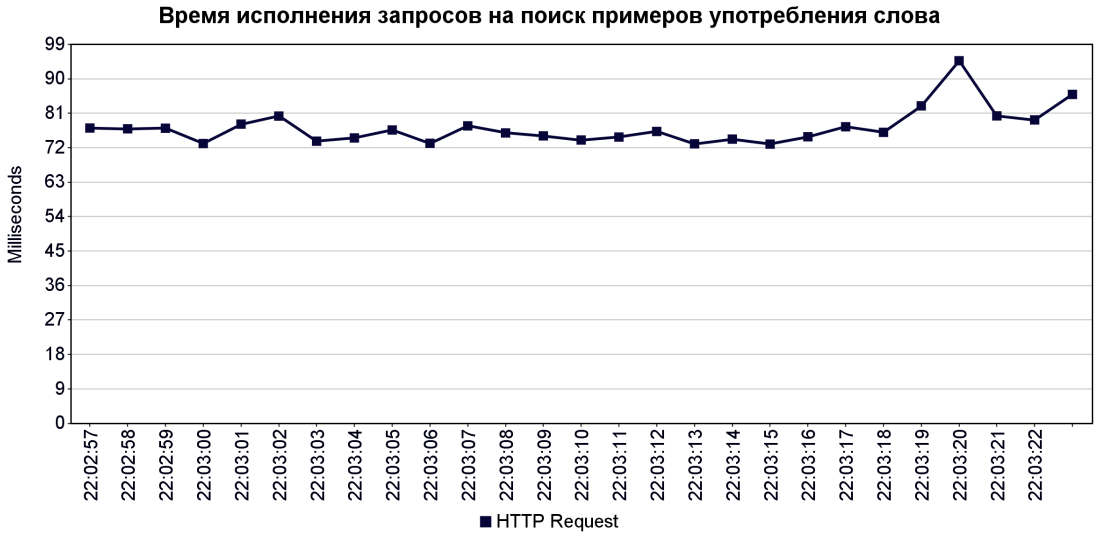

Рисунок 24 – Время выполнения «сложного» запроса к системе без использования кеширующей СУБД

Таким образом, благодаря использованию кеширующей СУБД, время получения ответа как на «простой», так и на «сложный» запрос уменьшается на 25%: с 80 мс до 60 мс. Однако, в случае использования кеширующей СУБД, при первом получении запроса необходимо записать значение в кеш.

Вывод

В результате проведенных замеров и, в том числе нагрузочного тестирования, установлено, что использование кеширующей СУБД позволяет сократить время повторного получения данных на 25%. Однако, в связи с необходимостью записи данных в кеширующую СУБД, время получения ответа на новый запрос увеличивается.

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была проанализирована предметная область параллельных корпусов текстов, были даны определения ключевым терминам и описаны способы применения корпуса. На основе проведенного анализа были сформулированы требования к наполнению корпуса и его возможностям, разработана ролевая модель. В качестве используемой разметки были выбраны морфологическая аннотация и метаразметка. В результате проведенного анализа существующих СУБД было выявлено, что наиболее подходящим для решения поставленной задачи видом СУБД является реляционная база данных, развернутая локально.

Были выделены сущности проектируемой базы данных, представлены ER-диаграмма в нотации Чена и диаграмма проектируемой базы данных. Для всех таблиц были описаны их атрибуты. Разработан алгоритм хранимой функции поиска примеров употребления слова в корпусе, а также алгоритм функции поиска с использованием фильтров. Кроме того, был описан формат кеширования данных и хранения логов.

В качестве реляционной СУБД была выбрана PostgreSQL, в качестве кеширующей СУБД Redis. Приложение к базе данных было написано на языке C# в формате Web API. Кроме того, в качестве вспомогательного сервиса к нему было написано приложение на языке Python, занимающееся обработкой лингвистических данных. Разработанная система логирования была реализована. Была протестирована ролевая модель.

В результате замеров времени и нагрузочного тестирования было установлено, что использование кеширующей СУБД сокращает время повторного получения данных на 25%. Однако, в связи с необходимостью записи данных в кеширующую СУБД, время получения ответа на новый запрос увеличивается.

Итак, решены следующие задачи:

— проанализирована предметная область параллельных корпусов текстов,

сформированы требования для хранения разметки текстов;

— проанализированы существующие СУБД;
— спроектирована БД для хранения и пополнения параллельного корпуса текстов и ролевая модель;
— спроектировано кеширование запросов с использованием дополнительной БД, спроектирована систему логирования действий пользователей;
— выбраны средства реализации системы, реализована спроектированная БД и необходимый интерфейс для взаимодействия с ней;
— реализована система логирования действий пользователей и проведено тестирование разделения ролей;
— проведено нагрузочное тестирование, определена зависимость времени исполнения запросов от использования кеширующей БД.

Таким образом, цель данной курсовой работы выполнена: разработана база данных для хранения и обработки параллельного корпуса переведённых текстов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ибатова, А.Ш. Определение и понятие лексемы в лексикологии / А.Ш. Ибатова // Science and world. – 2020. – P. 42-43.

2. Rura, L., Vandeweghe, W., Montero Perez, M. Designing a parallel corpus as multifunctional translator's aid // Proceedings of the 18th FIT world Congress. – 2008. – 11 p.

3. Мальцева, М.С. К определению термина «Лингвистический корпус» и его вариантов / М.С. Мальцева // Социально-экономические явления и процессы. – 2011. – №7. – С. 255-258.

4. Khosla S., Acharya H. A survey report on the existing methods of building a parallel corpus // International Journal of Advanced Research in Computer Science. – Vol.9, №4. – 2018. – P. 13-19.

5. Essential Speech and Language Technology for Dutch: Results by the STEVIN programme / P. Spyns [et al.] – The Hague: Springer, 2013. – 414 p.

6. Szudarski P. Corpus Linguistics for Vocabulary: A Guide for Research / P. Szudarski. – London: Routledge, 2017. – 238 p.

7. The Routledge Handbook of Corpus Linguistics / A. O'Keeffe [et al.] – London: Routledge, 2010. – 712 p.

8. Виды разметки – Национальный корпус русского языка. Режим доступа: https://ruscorpora.ru/page/annotation/ (дата обращения: 07.03.2023)

9. Leech G. Corpus Annotation Schemes // Literary and Linguistic Computing. – Vol. 8, №4. – 1993. – P. 275-281.

10. Archer D., Culpeper J., Davies M. Pragmatic Annotation // Corpus Linguistics: An International Handbook. – Berlin: Mouton de Gruyter, 2008. – P. 613-641.

11. Национальный корпус русского языка. Режим доступа: https://ruscorpora.ru/ (дата обращения: 07.03.2023)

12. Poibeau, T. Machine Translation / T. Poibeau. – Cambridge: the MIT Press, 2017. – 215 p.

13. Extensible Markup Language (XML). Режим доступа: https://www.w3.org/XML/ (дата обращения: 01.03.2023)

14. HTML Standard. Режим доступа: https://html.spec.whatwg.org/multipage/ (дата обращения: 01.03.2023)

15. Pęzik P., Ogrodniczuk M., Przepiórkowski A. Parallel and spoken corpora in an open repository of Polish language resources. – 2011. – 6 p.

16. Тао, Ю. Захаров, В.П. Разработка и использование параллельного корпуса русского и китайского языков // Информационные процессы и системы. – 2015.– №4. – С. 18-29.

17. Павлов, М.Н. Подходы к организации загрузки информации из xml-документов в реляционные базы данных // Известия Орловского Государственного Технического Университета. Серия: Информационные системы и технологии. – 2004. – №5(6). – С. 106-109

18. Кренке, Д. Теория и практика построения баз данных / Д. Кренке. – 9-е изд. – СПб.: Питер, 2005. – 859 с.

19. Карпова, И.П. Базы Данных. Учебное пособие / И. П. Карпова. – М.: Московский государственный институт электроники и математики (Технический университет), 2009. – 131 с.

20. Гущин, А.Н. Базы данных: учебно-методическое пособие / А. Н. Гущин. – 2-е изд., испр. и доп. – М.-Берлин: Директ-Медиа, 2015. – 311 с.

21. Аврунев, О.В., Стасышин, В.М. Модели баз данных: учебное пособие / О.Е. Аврунев, В.М. Стасышин. – Новосибирск: Изд-во НГТУ, 2018. – 124 с.

22. Шилин, А.С. Практическая нецелесообразность нормальных форм высокого порядка // Информатика и прикладная математика. – 2016. – №. 22. – С. 116-122.

23. Парфенов, Ю.П. Постреляционные хранилища данных : учеб. пособие / Ю.П. Парфенов. – Екатеринбург: Изд-во Урал. ун-та, 2016. – 120 с.

24. Reverso Context: использование контекстного переводчика. Режим доступа: https://context.reverso.net/перевод/about/ (дата обращения: 10.03.2023)

25. ABBYY Lingvo Live – онлайн-словарь. Режим доступа: https://www.lingvolive.com/ru-ru/ (дата обращения: 10.03.2023)

26. Linguee Dictionary. Режим доступа: https://www.linguee.com/ (дата обращения: 10.03.2023)

27. PostgreSQL: Documentation. Режим доступа: https://www.postgresql.org/docs/ (дата обращения: 15.04.2023)

28. Taipalus, T. Database Management System Performance Comparisons: A Systematic Survey / T. Taipalus. – Jyväskylä: University of Jyväskylä, 2023. – 32 p.

29. MySQL Documentation. Режим доступа: https://dev.mysql.com/doc/ (дата обращения: 15.04.2023)

30. SQL Server technical documentation. Режим доступа: https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16 (дата обращения: 15.04.2023)

31. CREATE FUNCTION Statement for Loadable Functions. Режим доступа: https://dev.mysql.com/doc/refman/5.7/en/create-function-loadable.html (дата обращения: 15.04.2023)

32. TIOBE Index for April 2023. Режим доступа: https://www.tiobe.com/tiobe-index/ (дата обращения: 15.04.2023)

33. Документация по C#. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 15.04.2023)

34. Python documentation. Режим доступа: https://docs.python.org/3/ (дата обращения: 15.04.2023)

35. Rider: Fast & powerful cross-platform .NET IDE. Режим доступа: https://www.jetbrains.com/rider/ (дата обращения: 15.04.2023)

36. PyCharm: IDE для профессиональной разработки на Python. Режим доступа: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения: 15.04.2023)

37. Introduction to Redis. Режим доступа: https://redis.io/docs/about/ (дата обращения: 15.04.2023)

38. Create web APIs with ASP.NET Core. Режим доступа: https://learn.microsoft.com/en-us/aspnet/core/web-api/?WT.mc_id=dotnet-35129-website&view=aspnetcore-7.0 (дата обращения: 15.04.2023)

39. Swagger Documentation. Режим доступа: https://swagger.io/docs/ (дата обращения: 15.04.2023)

40. NLog. Flexible & free open-source logging for .NET. Режим доступа: https://nlog-project.org/ (дата обращения: 15.04.2023)

41. Docker Docs: How to build, share and run applications. Режим доступа: https://docs.docker.com/ (дата обращения: 15.04.2023)

42. Wilkins, P. Logging in Action. With Fluentd, Kubernetes and more / Phil Whilkins. – Shelter Island: Manning, 2022. – 394 p.

43. Windows. Режим доступа: https://www.microsoft.com/ru-ru/windows (дата обращения: 15.04.2023)

44. Процессор Intel® Core™ i5-8265U. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/149088/intel-core-i58265u-processor-6m-cache-up-to-3-90-ghz.html (дата обращения: 15.04.2023)

45. Apache JMeter - Apache JMeter™. Режим доступа: https://jmeter.apache.org/ (дата обращения: 25.04.2023)

ПРИЛОЖЕНИЕ А

В листингах **??** – **??** приведены сценарии создания разработанных хранимых функций.

ПРИЛОЖЕНИЕ Б

Презентация к курсовой работе

Презентация содержит 17 слайдов.