МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт автоматики и информационных технологий

УДК _____ На правах рукописи

Айгожина Эльмира Нурлановна

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

На соискание академической степени магистра

Название диссертации	Разработка приложения с использованием методов семантической обработки
Направление подготовки	7M06101– Software Engineering

Научный руководитель
PhD, ассоциированный профессор
_____Сатыбалдиева Р.Ж.
«__» _____ 2023 г.

Оппонент
канд. физ-мат. наук, ст. преподаватель
_____И.М. Уалиева
«__» _____ 2023 г.

Нормоконтроль
PhD, ассиестент-профессор
_____ А.Т. Ахмедиярова
«__» _____ 2023 г.

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
канд. физ-мат. наук, проффессор
А.Н. Молдагулова
«__»_____ 2023 г.

Алматы 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Казахский национальный исследовательский технический
университет имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра Программная инженерия
Специальность: 7М06101– Software Engineering

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
канд. физ-мат. наук, проффессор
_____ А.Н. Молдагулова
«___ » _____ 2023 г.

ЗАДАНИЕ
на выполнение магистерской диссертации

Магистранту: Айгожиной Эльмире Нурлановной
Тема: «Разработка приложения с использованием методов семантической обработки»
Срок сдачи законченной диссертации «__» _____

Исходные данные к магистерской диссертации: Определена диссертационная работа по исследованию использования openAI для разработки системы тезауруса.

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации: а) создание приложения для поиска информации, в котором используются методы семантической обработки, в частности использование тезауруса;

б) подходы семантической обработки в разработке приложений, а также их эффективности в улучшении результатов поиска.

Рекомендуемая основная литература:

1) Christopher D., Prabhakar., Hinrich S. "Introduction to Information Retrieval" (2008)
2) Giovanni P., Giuseppe R., "Semantic Search Over the Web" (2007)
3) Jean A., Alan G., David B. "Thesaurus Construction and Use: A Practical Manual" (2001)

ГРАФИК
подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Раздел 1. Обзор по применению тезаурусов для поисковых систем	30.10.2022	Выполнено
Раздел 2. Подходы семантической обработки в разработке приложений	03.03.2023	Выполнено
Раздел 3. Практическая реализация	24.04.2023	Выполнено

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант, (уч. степень, звание)	Сроки	Подпись
Нормоконтроль	Ахмедиярова Айнур Танатаровна, PhD, ассоциированный профессор		
Программное обеспечение	Мукажанов Нуржан Какенович, ассоциированный профессор		
Антиплагиат	Аубакиров Бакдаулет, Ассистент		

Дата выдачи задания "____"_____ 2023 г.

Заведующий кафедрой _____Молдагулова А. Н.

Научный руководитель _____ Сатыбалдиева Р.Ж

Задание принял к исполнению магистрант _____ Айгожина Э. Н.

Дата "_____"_____ 2023 г.

АННОТАЦИЯ

Магистерская работа состоит из 4 глав, написанных на 63 страницах. Разработана система поиска информации на основе тезауруса.

Актуальность темы исследования: в настоящее время требуются более интеллектуальные и эффективные программные решения для обработки и анализа данных из-за сложности и разнообразия растущей сложности данных.

Область исследования: областью изучения этого исследования является информационные технологии, в частности, область информационного поиска и семантической обработки.

Объектом исследования: область семантической обработки и информационного поиска, включая разработку и применение семантических поисковых систем, и использование тезаурусов в информационном поиске.

Введение: Этот раздел представляет тезис, описывая определения основных терминов, подчеркивая важность проблемы исследования и указывая цель исследования, предпосылки, задачи и ожидаемые результаты.

В основной части диссертации обсуждаются различные темы, связанные с методами семантической обработки, их использование при разработке приложений, недостатки современного семантического поиска, обзор семантических поисковых систем, публикации о семантическом поиске, роль тезаурусов в поиске информации, описание эксперимента, дескрипторы информационно-поисковых тезаурусов, основные принципы разработки тезаурусов и специальные тезаурусы, такие как MeSH, APA и другие.

Анализ: В этом разделе основное внимание уделяется потребностям базы данных, в частности использованию SQLite в качестве технологии базы данных исследовательского проекта.

Практический раздел дипломной работы посвящен элементу реализации, включая создание базы данных, разработку интерфейса, описание программы и окончательный вариант интерфейса.

В целом, эта диссертация исследует многие элементы методов семантической обработки, их использование в поиске информации, а также создание и применение тезаурусов. Он включает в себя практическое применение и анализ, а также полное понимание предметной темы.

АҢДАТПА

Магистрдің жұмысы 4 тараудан тұрады, 63 бетте жазылған. Тезаурус негізінде ақпараттық іздеу жүйесі жасалды.

Зерттеу тақырыбының өзектілігі: қазіргі уақытта деректердің өсіп келе жатқан күрделілігінің күрделілігі мен әртүрлілігіне байланысты деректерді өңдеу және талдау үшін анағұрлым интеллектуалды және тиімді бағдарламалық шешімдер қажет.

Зерттеу саласы: Бұл зерттеудің зерттеу саласы ақпараттық технологиялар, атап айтқанда ақпаратты іздеу және семантикалық өңдеу саласы.

Зерттеу нысаны: семантикалық өңдеу және ақпаратты іздеу саласы, оның ішінде семантикалық іздеу жүйелерін жасау және қолдану және ақпаратты іздеуде тезауриді пайдалану.

Кіріспе: Бұл бөлімде негізгі терминдердің анықтамаларын сипаттай отырып, зерттеу мәселесінің маңыздылығына тоқталып, зерттеу мақсатын, алғышарттарын, міндеттерін және күтілетін нәтижелерді көрсету арқылы дипломдық жұмыс енгізіледі.

Дипломдық жұмыстың негізгі бөлімінде семантикалық өңдеу әдістеріне қатысты әртүрлі тақырыптар, олардың қолданбалы бағдарламаларды әзірлеуде қолданылуы, қазіргі семантикалық іздеудің кемшіліктері, семантикалық іздеу жүйелеріне шолу, семантикалық іздеу бойынша жарияланымдар, ақпарат іздеудегі тезауридің рөлі, мәтіннің мазмұнын сипаттау қарастырылады. эксперимент, ақпаратты іздеу тезаурисының дескрипторлары, тезауридің негізгі даму принциптері және MeSH, APA және т.б.

Талдау: Бұл бөлім дерекқордың қажеттіліктеріне, әсіресе SQLite-ті зерттеу жобасының дерекқор технологиясы ретінде пайдалануына бағытталған.

Дипломдық жұмыстың практикалық бөлімі енгізу элементіне арналған, оның ішінде мәліметтер қорын құру, интерфейсті өңдеу, бағдарламаның сипаттамасы және интерфейстің соңғы нұсқасы.

Тұтастай алғанда, бұл диссертация семантикалық өңдеу әдістерінің көптеген элементтерін, оларды ақпаратты іздеуде қолдануды және тезаурилерді құру мен пайдалануды зерттейді. Ол практикалық қолдану мен талдауды, сондай-ақ тақырыпты толық түсінуді қамтиды.

ANNOTATION

Master's work consists of 4 chapters, written on 63 pages. An information retrieval system based on thesaurus has been developed.

Relevance of the research topic: more intelligent and efficient software solutions for data processing and analysis are currently required due to the complexity and diversity of the growing complexity of data.

Field of study: The field of study of this study is information technology, in particular the field of information retrieval and semantic processing.

Object of research: the field of semantic processing and information retrieval, including the development and application of semantic search engines, and the use of thesauri in information retrieval.

Introduction: This section introduces the thesis by describing definitions of key terms, emphasizing the importance of the research problem, and indicating the research goal, background, objectives, and expected results.

The main part of the thesis discusses various topics related to semantic processing methods, their use in application development, shortcomings of modern semantic search, an overview of semantic search engines, publications on semantic search, the role of thesauri in information retrieval, description of the experiment, descriptors of information retrieval thesauri, basic thesauri development principles and specialized thesauri such as MeSH, APA and others.

Analysis: This section focuses on the needs of the database, specifically the use of SQLite as a research project database technology.

The practical section of the thesis is devoted to the implementation element, including the creation of the database, the development of the interface, the description of the program, and the final version of the interface.

Overall, this dissertation explores the many elements of semantic processing techniques, their use in information retrieval, and the creation and use of thesauri. It includes practical application and analysis, as well as a complete understanding of the subject matter.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящей диссертации применяют следующие термины с соответствующими определениями:

Семантический поиск — это процесс поиска неструктурированной информации, которую современные поисковые системы используют для предоставления наиболее релевантных результатов.

Термин происходит от названия раздела лингвистики «семантика», изучающего семантические значения [1].

Семантический поиск — это метод извлечения данных, в котором намерение и ресурсы пользователя представлены в семантической модели, то есть в иерархии между концепциями, а также во взаимосвязях (связности) между концепциями и сущностями [2].

Другими словами, семантический поисковый анализ направлен на нахождения запроса пользователя, с помощью контекстного значения и ключевых слов.

Существует логический поиск, который определяет нужную информацию и переводит на веб-страницу основываясь на ключевые слова. При это семантический поиск основывается на значении и контекста, что помогает расширить глубину информации и сопоставить страницы.

Тезаурус — это компьютерный словарь или база данных, включающая слова, которые похожи, противоположны, родственны и имеют другие семантические ассоциации. Тезаурусы используются для облегчения создания текста и повышения точности поиска информации в Интернете и в базах данных. Они могут быть полезными ресурсами для улучшения общения и расширения словарного запаса.

NLP - (Natural Language Processing) - это область исследований в области компьютерной лингвистики и искусственного интеллекта, которая занимается обработкой и анализом естественного языка, используемого людьми в повседневной жизни. Цель NLP - создать компьютерные системы, которые могут понимать и генерировать естественный язык, включая его лексику, грамматику, семантику и контекст [3].

Актуальность: Современные технологии не обходятся без разработки прикладного программного обеспечения, которое предлагает возможности, жизненно важные для многих аспектов повседневной жизни, включая социальные сети, администрирование компаний, электронную коммерцию, здравоохранение и образование. В настоящее время требуются более интеллектуальные и эффективные программные решения для обработки и анализа данных из-за сложности и разнообразия растущей сложности данных. Предлагая более сложный метод обработки естественного языка, методы семантической обработки были представлены как решение этих проблем, которое может значительно повысить производительность прикладного программного обеспечения. Чтобы создать интеллектуальные приложения, которые могут обрабатывать и анализировать данные более значимым образом,

эта диссертация посвящена изучению использования подходов семантической обработки в разработке программного обеспечения.

Исследование будет сосредоточено на цели создания прикладного программного обеспечения с использованием методов семантической обработки. В центре внимания исследования будет создание интеллектуальных приложений, которые могут обрабатывать и оценивать данные с использованием методов семантической обработки. Исследование будет следовать историческим рамкам, изучая, как методы семантической обработки менялись с течением времени и как они использовались для разработки программного обеспечения.

С акцентом на создание интеллектуальных приложений, которые могут обрабатывать и анализировать данные более значимыми способами, цель этой диссертации состоит в том, чтобы изучить использование подходов семантической обработки в разработке программного обеспечения.

Целью диссертации является создание программного обеспечения, использующего методы семантического поиска на основе тезауруса. Цель этой программы состоит в том, чтобы обрабатывать и анализировать данные более осмысленным образом, предоставляя потребителям более точные и актуальные выводы.

Цели:

1. Создание программного обеспечения с использованием методов семантического поиска на основе тезауруса. Основной целью диссертации является разработка программного обеспечения, использующего алгоритмы семантического поиска на основе тезауруса. Цель состоит в том, чтобы создать и развернуть систему, которая может улучшить поиск информации с помощью семантических ассоциаций, содержащихся в тезаурусе.

2. Улучшение обработки и анализа данных: цель программного обеспечения состоит в том, чтобы обрабатывать и анализировать данные более релевантным образом. Программа предназначена для предоставления клиентам более точной и актуальной информации за счет включения технологий семантического поиска на основе тезауруса. Это включает в себя повышение точности и полноты результатов поиска, а также более глубокое понимание взаимосвязей между понятиями и терминами.

3. Обеспечить всестороннее понимание семантического поиска и создания тезауруса: цель этого исследования — обеспечить полное понимание семантического поиска и создания тезауруса. Он направлен на изучение приложений, преимуществ и недостатков стратегий семантического поиска и использования тезаурусов в поиске информации. Цель состоит в том, чтобы дополнить текущий объем знаний в этом секторе и пролить свет на потенциал алгоритмов семантического поиска на основе тезауруса.

Исследование призвано расширить область семантического поиска и внести вклад в создание программного обеспечения, которое может улучшить анализ данных и процедуры поиска информации, достигнув этих целей. Результаты исследования предоставят информацию и практические выводы исследователям, разработчикам и потребителям, заинтересованным в

семантической обработке и использовании тезаурусов в различных дисциплинах.

Для достижения этой цели необходимо выполнить несколько задач. Сначала будет дано введение в методы семантической обработки и их использование в разработке программного обеспечения. Во-вторых, в диссертации будут рассмотрены трудности и ограничения, связанные с созданием интеллектуальных приложений с использованием методов семантической обработки. В-третьих, в исследовании будет рассмотрено, как методы семантической обработки влияют на эффективность, результативность и удобство использования программного обеспечения. На основе результатов исследования в диссертации будут предложены рекомендации и рекомендации по созданию интеллектуальных программных приложений.

Ожидаемым результатом этого исследования является углубление знаний об использовании методов семантической обработки в разработке программного обеспечения с упором на создание интеллектуальных приложений, которые могут обрабатывать и анализировать данные более осмысленным образом. Исследование предложит понимание трудностей и ограничений, связанных с созданием интеллектуальных приложений с использованием методов семантической обработки, а также предложит решения и передовой опыт для решения этих проблем. Разработчики программного обеспечения, ученые и другие заинтересованные стороны, заинтересованные в использовании методов семантической обработки в разработке программного обеспечения, сочтут выводы исследования полезными.

Для тестирования программы будут использоваться различные наборы данных, которые затем будут оцениваться по производительности, эффективности и пользовательскому опыту. Создание этого программного приложения послужит конкретным примером использования методов семантической обработки в разработке программного обеспечения, проливая свет на трудности и возможности, связанные с этой стратегией.

В частности, создание интеллектуального приложения, использующего стратегии семантического поиска на основе тезауруса, находится в центре внимания исследования этой диссертации по использованию методов семантической обработки в разработке программного обеспечения. Предоставляя более точные и релевантные результаты, цель состоит в том, чтобы повысить производительность, эффективность и удобство использования программного обеспечения. Подходы к семантической обработке будут рассмотрены в литературе, будут изучены их проблемы и ограничения, будет изучено их влияние на производительность программного обеспечения, и в рамках исследования будет разработано программное приложение для тестирования. Для проведения исследования будет использован ряд исследований, включая обзор литературы, создание программного приложения, тестирование и анализ результатов. Эта цель расширит знания о подходах к семантической обработке и их возможных преимуществах и недостатках

1. Основная часть

1.1 Описание методов семантической обработки

Понимая контекст и значение данных, ряд подходов, известных как методы семантической обработки, могут обрабатывать и анализировать данные более осмысленным образом. Семантика, изучение значения в языке и общении, является основой этих методов. Обработка естественного языка, интеллектуальный анализ данных и машинное обучение — это лишь некоторые из приложений, использующих методы семантической обработки.

Семантический поиск на основе тезауруса является одним из наиболее часто используемых подходов к семантической обработке. В этом методе используются тезаурусы или онтологии, которые представляют собой иерархические структуры, определяющие связи между различными терминами и понятиями. Определяя семантические связи между терминами, используемыми в запросе, и терминами, используемыми в данных, тезаурус используется для сопоставления запросов пользователей с соответствующими данными. Чтобы предоставить потребителям более точные и релевантные результаты, эта стратегия используется в поисковых системах и системах рекомендаций.

Распознавание именованных объектов (NER- named entity recognition), которое включает в себя идентификацию и категоризацию именованных объектов, таких как лица, организации и местоположения в текстовых данных, является еще одним методом семантической обработки. Приложения для обработки естественного языка, такие как чат-боты и виртуальные помощники, используют NER для понимания запросов пользователей и предоставления соответствующих ответов [4].

Структура описания ресурсов (RDF- Resource Description Framework), одна из технологий семантической паутины, используется для более осмысленного представления данных. Модель на основе графа используется RDF, стандартом представления онлайн-ресурсов, для определения отношений между элементами. RDF используется в таких приложениях, как управление знаниями и интеграция данных, чтобы обеспечить более полное и релевантное представление данных.

Семантическая обработка также использует методы глубокого обучения, такие как сверточные нейронные сети (CNN- convolutional neural networks) и рекуррентные нейронные сети (RNN- recurrent neural networks). Эти методы используют нейронные сети для обработки и анализа данных, что позволяет обнаруживать сложные закономерности и связи [5].

В заключение, понимая контекст и значение данных, методы семантической обработки представляют собой ряд подходов, используемых для более осмысленной обработки и анализа данных. Методы глубокого обучения, распознавание именованных сущностей, семантические веб-технологии и

семантический поиск на основе тезауруса — вот некоторые из этих подходов. Эти методы широко используются во многих различных отраслях промышленности и постоянно меняются благодаря достижениям науки и техники.

1.2 Использование методов семантической обработки

Поскольку они могут давать более точные и релевантные результаты, методы семантической обработки становятся все более и более важными в различных областях. Приложения для этих методов включают поиск информации, машинное обучение, интеллектуальный анализ данных и обработку естественного языка.

Поисковые системы — одно из мест, где чаще всего используются методы семантической обработки. Определяя семантические связи между терминами, используемыми в запросе, и терминами, используемыми в данных, семантический поиск на основе тезауруса помогает пользователям получать более точные и релевантные результаты. Поисковые системы могут понять намерение пользователя, используя эту стратегию, и предоставить более подходящие результаты [6].

Например, если пользователь ищет «яблоко», поисковая система, которая понимает контекст запроса, может определить разницу между фруктами и технологическим бизнесом.

Чтобы дать потребителям индивидуальные предложения, основанные на их предпочтениях и предшествующем поведении, системы рекомендаций также используют методы семантической обработки. Эти системы отслеживают поведение и предпочтения пользователей с помощью алгоритмов машинного обучения и находят нужные вещи с помощью семантического поиска на основе тезауруса.

Методы семантической обработки используются в области обработки естественного языка для расшифровки значения текстовых данных. Чтобы понять контекст текста, распознавание именованных объектов используется для поиска и классификации именованных объектов в текстовых данных. Некоторые приложения, в том числе мониторинг социальных сетей и обслуживание клиентов, используют анализ тональности, который использует алгоритмы машинного обучения для оценки тональности текстовых данных.

Для более тщательного и релевантного отображения данных в управлении знаниями и интеграции данных используются методы семантической обработки.

1.3 Методы семантической обработки при разработке приложений

Поскольку они могут давать более точные и релевантные результаты, важность подходов семантической обработки в разработке приложений возросла. С помощью этих методов можно создавать приложения, которые понимают и интерпретируют контекст и значение данных, обеспечивая более интеллектуальное и естественное взаимодействие с пользователями.

Приложения обработки естественного языка (NLP) являются одним из наиболее часто используемых контекстов для методов семантической обработки при разработке приложений. Программное обеспечение NLP использует методы семантической обработки, такие как идентификация именованных объектов и анализ настроений, для интерпретации ввода текста и осмысленного ответа на запросы пользователя. Виртуальные помощники и чат-боты — два примера систем NLP, которые используют семантические подходы для понимания запросов пользователей и предоставления соответствующих ответов [7].

Системы выдачи рекомендаций при разработке приложений также используют методы семантической обработки. Эти системы предоставляют клиентам индивидуальные предложения, основанные на их интересах и предшествующем поведении, с использованием алгоритмов машинного обучения и семантического поиска на основе тезауруса. Например, система музыкальных рекомендаций может использовать методы семантической обработки для определения жанра и настроения предпочитаемых пользователем песен, чтобы предлагать треки, похожие на эти любимые.

При создании приложений поисковые системы также используют методы семантической обработки. Эти поисковые системы понимают контекст пользовательских запросов и предоставляют более релевантные результаты, используя семантический поиск на основе тезауруса. Это позволяет разработчикам приложений создавать поисковые системы, которые могут предоставлять потребителям более точные и релевантные результаты.

Методы семантической обработки часто используются при разработке приложений для управления знаниями и интеграции данных. Эти методы позволяют комбинировать данные из нескольких источников, придавая им более полное и глубокое представление [8]. Это может быть полезно для создания таких программ, как приложения бизнес-аналитики и системы управления взаимоотношениями с клиентами, которым требуется доступ к большому количеству данных из многих источников.

Следующие исследовательские подходы могут быть использованы для изучения темы семантической обработки и разработки тезаурусов:

1) Проведение тщательного обзора существующей литературы и научных статей по семантической обработке, семантическому поиску и созданию тезаурусов. Это помогает получить полное представление о текущем состоянии области, выявить пробелы в исследованиях и направить дизайн исследования.

2) Сбор данных влечет за собой сбор соответствующих данных, таких как текстовые корпуса, исследовательские статьи или вводимые пользователем

данные, для изучения и оценки эффективности алгоритмов семантической обработки и тезаурусов. Это может включать получение данных из существующих баз данных, проведение опросов или интервью, или разработку наборов данных специально для темы исследования.

3) Анализ данных: изучение собранных данных и получение соответствующих выводов с использованием соответствующих инструментов анализа данных. Статистический анализ, интеллектуальный анализ текста или методы обработки естественного языка могут использоваться для исследования закономерностей, корреляций и тенденций в данных.

4. Экспериментирование — это процесс разработки и проведения экспериментов для проверки производительности и полезности разработанного программного обеспечения или технологии. Это может повлечь за собой разработку тестовых сценариев, определение мер оценки и сравнение результатов с базовыми или существующими процедурами для анализа воздействия и преимуществ предлагаемых подходов.

5. Прототипирование: создание прототипов программного обеспечения или приложений, использующих методы семантической обработки и тезаурусы. Это влечет за собой использование компьютерных языков, сред разработки программного обеспечения и инструментов для запуска намеченной системы или функции.

6. Пользовательские исследования: сбор информации о созданном программном обеспечении или тезаурусе посредством пользовательских исследований или тестирования удобства использования. Это помогает понять взгляды пользователей, найти места для развития и оценить удовлетворенность пользователей и производительность при использовании системы.

7. Сравнительный анализ: сравнение различных методов семантической обработки, тезаурусных моделей или поисковых систем для оценки их сильных и слабых сторон и применения в различных областях. Сравнительные эксперименты, оценка производительности и критический анализ результатов являются частью этого процесса.

Используя эти исследовательские подходы, исследование надеется дополнить существующий объем знаний, предоставить эмпирические данные и способствовать пониманию и использованию методов семантической обработки и тезаурусов в поиске информации и связанных областях.

Используя эти методы исследования, исследование направлено на то, чтобы внести свой вклад в существующую совокупность знаний, предоставить эмпирические данные и улучшить понимание и применение методов семантической обработки и тезаурусов в информационном поиске и смежных областях.

1.4 Недостатки современного семантического поиска

Хотя методы семантической обработки становятся все более и более важными при разработке приложений, у них есть существенные недостатки, которые необходимо учитывать. Зависимость современного семантического поиска от методов машинного обучения является одним из его основных недостатков. Эти алгоритмы требуют больших объемов обучающих данных, получить которые в некоторых обстоятельствах может быть сложно. Кроме того, предвзятые алгоритмы машинного обучения могут давать неверные результаты или усиливать предвзятость, уже присутствующую в данных.

Сложность ведения тезауруса — еще один недостаток современного семантического поиска. Чтобы отражать лингвистические изменения и новые идеи, тезаурус необходимо часто обновлять, что может быть трудоемкой и дорогостоящей операцией [9,10].

Семантический поиск также может требовать значительных вычислительных ресурсов, особенно при работе с огромными объемами данных. Приложения, которым необходимо быстро или в режиме реального времени анализировать большие объемы данных, могут столкнуться с трудностями.

Возможность неправильной интерпретации данных является еще одним недостатком семантического поиска. В основе методов семантической обработки лежит обнаружение сложных и нюансированных семантических связей между понятиями. Это может привести к ошибкам или неправильным представлениям при интерпретации данных, которые могут иметь серьезные последствия в определенных ситуациях.

И последнее, но не менее важное: применение методов семантической обработки при разработке приложений поднимает вопросы конфиденциальности и безопасности [11,12]. Эти методы основаны на доступе к огромным объемам данных, некоторые из которых могут быть частными или конфиденциальными. Во избежание нежелательного доступа или неправомерного использования крайне важно обеспечить конфиденциальность и безопасность этих данных.

Методы семантической обработки предоставляют много преимуществ для разработки приложений, но они также имеют некоторые недостатки, которые необходимо учитывать. К ним относятся потребность в методах машинного обучения, сложность поддержания тезауруса в актуальном состоянии, стоимость вычислений, возможность неправильной интерпретации данных, а также проблемы с конфиденциальностью и безопасностью. Чтобы разработчики могли создавать эффективные и ответственные приложения, использующие методы семантической обработки, они должны знать об этих недостатках.

1.5 Обзор семантических поисковых систем

Многие программные приложения были разработаны с использованием методов семантической обработки, особенно в области управления знаниями и поиска информации. Существующие семантические поисковые системы, включая Knowledge Graph от Google, Wolfram Alpha, IBM Watson, GPT-3 от OpenAI и Sibirix от Яндекса, иллюстрируют, насколько хорошо эти методы работают, предоставляя пользователям точные и исчерпывающие результаты поиска (10). Однако у этих поисковых систем есть несколько недостатков, таких как трудности с обработкой огромных и разнообразных наборов данных, предвзятость алгоритмов и трудности с интерпретацией языка и контекста.

Методы семантической обработки продолжают использоваться для создания новых приложений, несмотря на их ограничения из-за их огромных перспектив в разработке программных приложений. Препятствия, вызванные этими ограничениями, должны быть устранены для создания более сложных и объективных семантических поисковых систем, что необходимо для полного использования возможностей методов семантической обработки. Преимущества и недостатки использования методов семантической обработки при создании приложений будут подробно рассмотрены в следующих частях вместе с предложениями по решению этих проблем.

Популярность семантических поисковых систем выросла за последние несколько лет в результате их способности собирать и упорядочивать данные путем понимания контекста и значения поисковых запросов. Например, Knowledge Graph от Google, который был представлен в 2012 году и с тех пор превратился в мощный механизм поиска информации. Согласно исследованию 2017 года, опубликованному в Журнале Ассоциации информационных наук и технологий, предоставление потребителям более актуальной информации об их поисковых запросах значительно повысило точность результатов поиска.

В таблице 1.1 приведена разница между Wolfram Alpha и Knowledge Graph от Google.

Таблица 1.1

Сравнительный анализ поисковых систем Knowledge Graph от Google и Wolfram Alpha

	Knowledge Graph	Wolfram Alpha
Фокус	Knowledge Graph от Google ориентирован на предоставление информации о сущностях (например, людях, местах и вещах) и их отношениях.	Wolfram Alpha в первую очередь ориентирован на предоставление компьютерных ответов на вопросы, тогда как

Глубина информации	Knowledge Graph от Google предоставляет широкий спектр информации о сущностях, но может не обеспечивать такого же уровня глубины или интерактивности.	Wolfram Alpha предоставляет подробную информацию по широкому кругу тем, включая математику, естественные науки, финансы и многое другое. Он также обеспечивает интерактивную визуализацию и позволяет выполнять сложные вычисления.
Источники	Knowledge Graph от Google в основном опирается на общедоступные источники, такие как Wikipedia и Freebase.	Wolfram Alpha опирается на широкий спектр источников, включая тщательно отобранные базы данных и собственные алгоритм.
Пользовательский интерфейс	График знаний Google интегрирован в поисковую систему Google и представляет информацию в более традиционном формате результатов поиска, а дополнительная информация отображается в правой части экрана.	Wolfram Alpha имеет простой и понятный интерфейс с акцентом на предоставление точных ответов на конкретные вопросы.

Wolfram Alpha, запущенная в 2009 году, является хорошо известной семантической поисковой системой, которая часто используется для анализа данных и научных исследований.

Исследование 2019 года, опубликованное в International Journal of Web Information Systems, показало, что Wolfram Alpha работает лучше, чем обычные поисковые системы, при обработке сложных запросов. Когда он был представлен в 2013 году, IBM Watson также вызвал большой интерес благодаря своей способности анализировать огромные объемы неструктурированных данных и предлагать потребителям полезную информацию. Было обнаружено, что Уотсон способен точно диагностировать и рекомендовать лечение больных раком в исследовании 2017 года, которое было опубликовано в Журнале Американской ассоциации медицинской информатики [13,14,15].

SemMedDB, созданная Национальным центром биотехнологической информации (NCBI), превратилась в популярную семантическую поисковую систему для изучения биомедицинской литературы в области биомедицинских исследований. Согласно исследованию 2018 года, опубликованному в Journal of Biomedical Informatics, SemMedDB смогла точно определить существенные

связи между медицинскими концепциями и предоставила исследователям полный инструмент для изучения биомедицинской литературы.

С момента своего выпуска в 2010 году ElasticSearch приобрел известность как мощная поисковая система, использующая методы машинного обучения для понимания контекста и значения поисковых запросов [15]. ElasticSearch превзошел стандартные поисковые системы в исследовании, опубликованном в Journal of Web Engineering в 2018 году, когда речь шла о получении подходящего материала из огромных наборов данных.

IBM Watson, запущенный в 2011 году, представляет собой мощную семантическую поисковую систему, которая использует машинное обучение и обработку естественного языка для оценки неструктурированного материала и предоставления пользователям подходящей информации. У Watson есть несколько применений, особенно в области здравоохранения, финансов и обслуживания клиентов.

В мире семантических поисковых систем появились более поздние участники, такие как OpenAI , который был представлен в июне 2020 года. Он считается значительным достижением в области искусственного интеллекта, поскольку он использует машинное обучение для получения ответов на естественном языке на запросы пользователей.

Другой известной семантической поисковой системой, которая использует семантический анализ для предоставления потребителям релевантных результатов поиска на русском языке, является Sibirix от Яндекса, выпущенная в 2019 году [16]. В ближайшие годы мы можем ожидать появления еще более мощных и интеллектуальных поисковых систем из-за продолжающегося роста. и инновации в области семантической обработки.

Эти семантические поисковые системы предоставляют множество функций и возможностей для удовлетворения различных требований и предпочтений пользователей. Мы можем ожидать появления все более совершенных и творческих поисковых систем, поскольку дисциплина семантической обработки продолжает развиваться, революционизируя то, как мы получаем доступ к информации и обрабатываем ее.

1.6 Публикации о семантическом поиске.

Количество работ, посвященных тематике семантического поиска, за последние годы значительно выросло. В этих книгах рассматривается широкий круг тем, в том числе машинное обучение, обработка естественного языка и поиск информации.

Следует отметить статью Ахмеда Аббаси и Хсинчуна Чена «Обзор подходов к семантическому поиску» в журнале ACM Transactions on Information Systems за 2008 год. Это исследование предлагает тщательный обзор различных методологий семантического поиска и секторов, в которых они применяются.

Авторы подчеркивают необходимость дополнительных исследований в этой области, обсуждая ценность семантического поиска в устранении недостатков обычных поисковых систем, основанных на ключевых словах.

Статья Дженса Граупмана, Томаса Франца и Сёрена Ауэра в журнале Web Semantics за 2012 год «SemSearch: поисковая система для семантической сети» — еще одна заслуживающая внимания работа [17]. В этой работе описывается SemSearch, поисковая система, использующая семантические технологии для поиска ресурсов в Semantic Web. Авторы проводят тщательный анализ производительности SemSearch в дополнение к описанию его архитектуры и возможностей.

Суджата Панди, Сушил Кумар и Браджеш Кумар Сингх опубликовали статью «Последние достижения в технологиях семантического поиска» в Международном журнале управления информацией в 2018 году. В этой статье освещаются самые последние достижения в области графов знаний, глубокого обучения и обработки естественного языка в их отношении. к технологии семантического поиска.

В целом, эти статьи показывают, насколько важным и популярным становится семантический поиск в области поиска информации. Они предлагают полезную информацию о различных методах и инструментах, используемых в семантическом поиске, и их потенциальном применении в различных областях. Семантический поиск становится все более важным инструментом для быстрого и точного поиска необходимой информации, поскольку объем и сложность данных продолжают расти.

1.7 Тезаурус в информационном поиске.

Надежным методом повышения эффективности и действенности процесса поиска информации является использование тезауруса. Тезаурус — это регулируемый словарь, в котором есть список слов и то, как они соотносятся друг с другом. Для поиска информации он предлагает единый и стандартизированный метод описания понятий и их отношений. В этой части мы рассмотрим использование тезауруса в информационном поиске.

Повышение точности и памяти результатов поиска является одной из основных целей тезауруса при поиске информации. Использование контролируемого словаря позволяет отображать поисковые термины в синонимы в тезаурусе, что может помочь устранить двусмысленность и повысить точность поиска.

Например, поиск по слову «автомобиль» может дать информацию об автомобилях, тогда как поиск по слову «транспортное средство» может дать информацию как о легковых автомобилях, так и о грузовиках. Процедуру поиска можно улучшить, связав оба этих поисковых запроса с одной и той же фразой тезауруса [18]. Тезаурусы также можно использовать для облегчения поиска и навигации по огромным информационным ресурсам. Пользователи могут просто

перемещаться между связанными понятиями и исследовать отношения между ними, группируя понятия в иерархическую структуру. В результате пользователи могут изучать новый материал и лучше понимать сложные предметы.

Тезаурусы также можно использовать для автоматической индексации и классификации материала. Процесс поиска можно улучшить, используя фразы из тезауруса в качестве ключевых слов индексации для получения более точных и эффективных результатов. Кроме того, материал можно автоматически классифицировать по различным предметным областям путем сопоставления понятий с соответствующими категориями тезауруса, что может помочь улучшить управление и организацию обширных информационных коллекций. Тезаурусы могут использоваться в поисковых системах для расширения поиска пользователей и предоставления альтернативных терминов, которые могут лучше подходить для их поиска. Это может увеличить полноту и точность результатов поиска, упростив потребителям поиск того, что они ищут.

Термин «банк», который может использоваться для описания как финансовой организации, так и берега реки, является одним из примеров запроса, который имеет множество значений и может быть решен с помощью тезауруса. Тезаурусы можно использовать для стандартизации терминологии, используемой для определения различных тем или предметов в текстовой классификации [19,20]. Даже если в статьях используется разная лексика, это может помочь обеспечить их единообразную классификацию. Для организации и структурирования больших коллекций документов можно также использовать тезаурус для поиска схожих идей и тем.

Тезаурус можно использовать при анализе текста для поиска и извлечения соответствующей информации из неструктурированных текстовых данных. Можно найти связанные понятия и собрать данные об их взаимосвязях, сопоставив текст с понятиями в тезаурусе. При выполнении таких задач, как распознавание сущностей и устранение неоднозначности сущностей, тезаурус можно использовать для поиска и извлечения именованных сущностей, таких как лица, местоположения и организации.

В целом было продемонстрировано, что использование тезауруса при поиске информации повышает релевантность и точность результатов поиска, упрощая пользователям поиск необходимой им информации. Использование тезаурусов в информационном поиске, по прогнозам, будет становиться все более важным в ближайшие годы из-за растущей доступности крупномасштабных тезаурусов и инструментов семантической обработки.

1.8 Дескрипторы информационно-поискового тезауруса

Эффективные системы поиска информации должны иметь тезаурусные описания для поиска информации. По сути, они представляют собой набор слов, которые используются для характеристики идей и предметов, охватываемых

конкретной базой данных или источником информации. Легче идентифицировать соответствующие ресурсы благодаря стандартизированному языку, которому способствуют эти дескрипторы.

Точность и полнота информационно-поисковых систем могут быть значительно повышены за счет использования тезаурусных описаний, что дает более точные и релевантные результаты поиска. Проблема синонимов и вариантов терминологии, которые могут привести к потере информации или нерелевантным результатам, решается с использованием дескрипторов тезауруса. Дескрипторы тезауруса могут помочь в раскрытии предметов и понятий в дополнение к стандартизации терминологии. Они предлагают основу для изучения тематической области и позволяют пользователям перемещаться между более общими и подробными понятиями. Пользователи, которые плохо знакомы с определенной предметной областью, могут найти это особенно полезным, поскольку это может дать им основу для понимания и исследования предмета.

Дескрипторы тезауруса также широко используются при индексировании документов, когда документу присваивается описательная фраза для ускорения поиска. Поскольку дескрипторы предлагают единый и стандартизированный способ описания содержимого документа, их применение может повысить точность индексации. Техника контролируемой лексики обычно используется для разработки дескрипторов тезауруса. Группа экспертов в предметной области находит и выбирает лучшие термины для описания понятий в определенной теме [21,22]. Затем в тезаурусе эти термины располагаются иерархически, где более широкие термины связаны с более узкими, а родственные фразы связаны друг с другом.

Одно из преимуществ использования дескрипторов тезауруса для поиска информации заключается в том, что они предлагают единый словарь для определения понятий, который может повысить запоминаемость и точность результатов поиска. Рекомендуя связанные фразы и более широкие или узкие понятия, они также могут помочь пользователям в изучении новых идей, связанных с их поисковым запросом.

Однако создание и поддержание тезауруса может занять некоторое время и потребовать специальных знаний. Кроме того, не все типы информационного поиска могут выиграть от использования дескрипторов тезауруса, поскольку для некоторых поисковых запросов может потребоваться более гибкий и разнообразный метод индексации и поиска.

В целом, эффективность информационно-поисковых систем может быть значительно повышена за счет включения дескрипторов тезауруса. Они предлагают единый стандартизированный словарь для описания информации, помогают преодолевать проблемы с синонимами и терминологическими различиями, облегчают идентификацию связанных понятий и повышают точность индексации документов.

1.9 Основные принципы разработки тезаурусов

Организация и поиск информации в различных областях, таких как информатика, библиотечное дело и управление данными, зависят от тезаурусов. Чтобы обеспечить полезность и актуальность тезауруса для предполагаемых пользователей, во время разработки следует использовать методическую процедуру, которая должна соответствовать нескольким фундаментальным принципам.

Определение объема и цели тезауруса является одним из основных процесса. На этом шаге необходимо указать тему или область, которую будет охватывать тезаурус, а также целевой рынок или пользователей. Объем и цель тезауруса служат руководством для выбора соответствующих терминов, понятий и отношений для включения. Также важность в создании информационно-поисковых тезаурусов является выбор терминов для включения в тезаурус. Существуют различные источники терминологии, которые можно использовать для создания тезаурусов поиска информации. В первую очередь следует изучить существующие тезаурусы по сходным предметным областям. Они могут содержать большое количество ценных терминов для нового тезауруса. Эксперты в предметной области могут рекомендовать термины для включения в тезаурус. Кроме того, термины тезауруса могут быть извлечены из тематических статей с помощью автоматической или ручной обработки документов. Индексаторы вручную анализируют документы, сначала индексируя поступающие статьи по наиболее важным ключевым словам, которые затем объединяются в единый список, который может служить основой для тезауруса (Архангельская, Базарнова, 2001) [23]. Тезаурус должен содержать термины, которые являются конкретными, однозначными и имеют отношение к данной области. Используя более широкие термины, более узкие термины и связанные термины, термины должны быть сгруппированы таким образом, чтобы представлять их отношения и иерархии. Также важно убедиться, что термины соответствуют языку и терминологии предполагаемых потребителей.

После того, как список терминов-кандидатов составлен, термины, которые появляются слишком часто, удаляются, поскольку считается, что они не предоставляют достаточной информации для различения конкретных текстов. Термины с низкой частотой встречаемости могут быть удалены из списка или предоставлены в качестве дескрипторов более общих или часто встречающихся понятий.

Третий принцип заключается в поддержании согласованности и качества тезауруса на высоком уровне. Это влечет за собой обновление и переписывание тезауруса на регулярной основе, чтобы гарантировать, что он остается актуальным и актуальным с учетом изменений в предмете, который он охватывает. Тезаурус должен быть проверен и обновлен на основе отзывов пользователей и профильных экспертов. Поддержание согласованности также означает соблюдение установленных стандартов и принципов разработки

тезауруса и обеспечение правильности и согласованности отношений и иерархий между терминами.

Наконец, обеспечение доступности и пригодности тезауруса для предполагаемых пользователей является основной концепцией создания тезауруса. Сюда входит создание удобного для пользователя тезауруса с использованием четких и лаконичных формулировок и предоставление инструкций по эффективному использованию тезауруса.

Подводя итог, можно сказать, что разработка тезауруса требует соблюдения некоторых основных критериев, чтобы гарантировать его полезность и актуальность для предполагаемых потребителей. Эти принципы включают определение объема и назначения тезауруса, выбор и организацию терминов, сохранение согласованности и качества, а также обеспечение доступности и удобства использования тезауруса для пользователей.

Разработчики тезауруса LIV Исследовательской службы Конгресса США (LIV, 1994) описывают правила включения терминов в тезаурус следующим образом:

- термины тезауруса должны представлять понятия, которые реально упоминаются в литературе, и должны отбираться из соображений эффективности их использования в поиске документов; - важным фактором включения термина является частотность его упоминания в текстах, которую необходимо периодически проверять;

- включение новых терминов в тезаурус должно происходить с учетом уже включенных тезаурусных терминов. Термины-кандидаты должны проверяться на предмет соответствия их общности/специфичности к другим терминам тезауруса. Также должно проверяться, представляет ли термин-кандидат отдельное понятие, которому нет соответствий среди существующих терминов тезауруса. Необходимо избегать включения терминов, чьи значения пересекаются со значениями уже существующих тезаурусных терминов настолько, что индексаторам и пользователям будет трудно различать между ними [24].

Таким образом, разработка хорошего информационно-поискового тезауруса представляет собой достаточно сложный, многоэтапный процесс, в котором необходимо найти «золотую середину». С одной стороны, набор дескрипторов тезауруса должно быть достаточен для описания произвольного документа предметной области, с другой стороны, дескрипторов не должно быть слишком много, поскольку слишком большая величина тезауруса повышает субъективность индексирования и затрудняет развитие и использование тезауруса. Не случайно, значительная доля информационно-поисковых тезаурусов в самых широких областях включает не более 10 тысяч терминов и 6-7 тысяч дескрипторов. Широко известным исключением являются Тезаурус по архитектуре и искусству (Тезаурус AAT), содержащий более 30 тысяч дескрипторов, что, видимо, связано со спецификой соответствующей предметной области, когда нужно индексировать не столько документы, сколько конкретные музейные предметы. Другим известным исключением, сверхбольшим тезаурусом является тезаурус по медицине MeSH, что связано с

гетерогенностью области медицины, состоящей из множества подобластей с собственной терминологией [25].

1.10 Конкретные тезаурусы

Существуют различные научные тезаурусы, которые регулярно используются благодаря их широкому охвату и признанному авторитету. Эти тезаурусы необходимы для организации и поиска научных данных.

1.10.1 MeSH

MeSH — хорошо известный и широко используемый тезаурус в области биомедицины и наук о жизни. Ее создала и поддерживает Национальная медицинская библиотека (NLM) в США. MeSH предлагает определенный словарь терминов для индексирования и поиска биологической литературы, обеспечивая точный и последовательный поиск соответствующей информации.

MeSH создается с помощью строгого и повторяющегося процесса, который обеспечивает его точность, актуальность и адаптируемость. Вот некоторые технические особенности разработки MeSH:

Идентификация концепции: Эксперты в предметной области и библиотекари определяют и определяют биомедицинские концепции. Эти термины обозначают многие аспекты болезней, лечения, анатомии, веществ и других тем.

Выбор термина: для каждого понятия определяется набор терминов и синонимов. Терминология, выбранная для включения в MeSH, выбрана с осторожностью, чтобы быть репрезентативной, широко используемой и стандартизированной. Они проверяются и проверяются, чтобы гарантировать их точность и уместность [26].

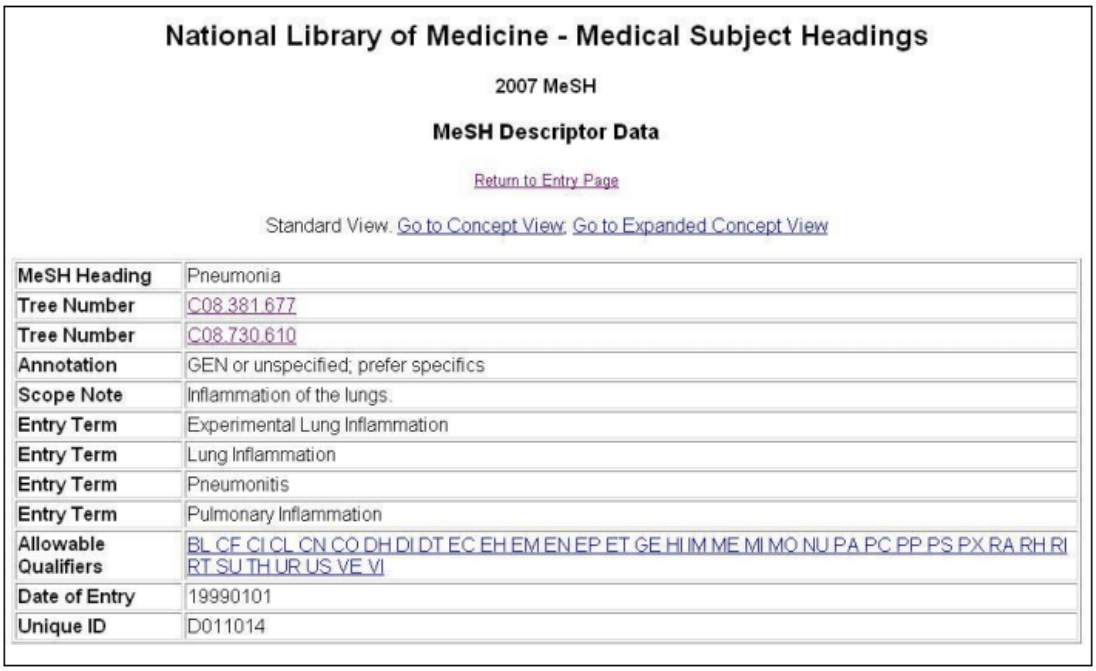
Иерархическая организация: MeSH организует концепции с использованием иерархической структуры. Он использует древовидную структуру, в которой более широкие термины (понятия более высокого уровня) связаны с более узкими терминами (понятиями более низкого уровня).

Формирование отношений: в дополнение к иерархическим отношениям MeSH формирует дополнительные типы отношений между терминами, такие как связанные термины и синонимичные термины. Эти соединения улучшают точность и память результатов поиска, предоставляя пользователям альтернативные или связанные термины для изучения [27].

MeSH постоянно обновляется и поддерживается, чтобы не отставать от развивающихся биомедицинских знаний и словаря. Вводятся новые понятия и термины, а устаревшие или более не актуальные термины удаляются или модифицируются. Экспертная оценка, обратная связь с пользователями и

рассмотрение новых научных результатов и улучшений — все это часть непрерывного процесса обслуживания.

MeSH в основном разработан на английском языке, хотя он также включает переводы многих терминов на другие языки. Эта многоязычная возможность повышает доступность и удобство использования MeSH для исследователей.



National Library of Medicine - Medical Subject Headings

2007 MeSH

MeSH Descriptor Data

Return to Entry Page

Standard View: Go to Concept View; Go to Expanded Concept View

MeSH Heading	Pneumonia
Tree Number	C08.381.677
Tree Number	C08.730.610
Annotation	GEN or unspecified; prefer specifics
Scope Note	Inflammation of the lungs.
Entry Term	Experimental Lung Inflammation
Entry Term	Lung Inflammation
Entry Term	Pneumonitis
Entry Term	Pulmonary Inflammation
Allowable Qualifiers	BL CF CI CL CN CO DH DI DT EC EH EM EN EP ET GE HI IM ME MI MO NU PA PC PP PS PX RA RH RI RT SU TH UR US VE VI
Date of Entry	19990101
Unique ID	D011014

Рисунок 1. Словарная статья тезауруса MeSH

Показана словарная статья дескриптора воспаление легких (рисунок1). Данный дескриптор относится к двум иерархическим деревьям – одно дерево легочные заболевания, второе – респираторные инфекции.

MeSH стал важнейшим компонентом систем индексирования и поиска биологической литературы, таких как PubMed, база данных популярных научных статей. MeSH используется исследователями, врачами и библиотекарями для точного и тщательного поиска, что позволяет им находить соответствующие публикации и эффективно исследовать конкретные биомедицинские темы.

MeSH создан в результате сотрудничества экспертов в данной области, специалистов по информации и строгих систем контроля качества. Его постоянное развитие гарантирует, что он будет по-прежнему оставаться жизненно важным ресурсом для организации и поиска биомедицинской информации, способствуя прогрессу в медицинских исследованиях и клинической практике [27].

1.10.2 Тезаурус APA

Тезаурус терминов психологического индекса (тезаурус APA): Тезаурус APA — бесценный ресурс для психологов, исследователей и профессионалов. Его создала Американская психологическая ассоциация (АПА), и он охватывает широкий круг тем и слов, относящихся к психологии и смежным наукам. Тезаурус АПА, который предоставляет стандартизированный язык для описания психологических понятий, часто используется для индексирования и поиска литературы по психологии.

Ниже приведены некоторые важные особенности тезауруса APA:

Охват понятий и фраз: Тезаурус АПА охватывает широкий спектр понятий и фраз, относящихся к психологии и смежным наукам. Он включает слова из нескольких разделов психологии, включая когнитивную психологию, психологию развития, клиническую психологию, социальную психологию и другие. Теории, методы, психологические расстройства, психологические обследования и терапевтические вмешательства входят в число тем, охватываемых тезаурусом [28].

Выбор терминов и стандартизация. Для обеспечения единообразия и точности термины, включенные в тезаурус APA, тщательно отобраны и стандартизированы. В процессе отбора используется экспертная обратная связь от психологов и информационных специалистов. Термины были выбраны таким образом, чтобы надлежащим образом представлять концепции в области психологии, а также отражать современный язык и понимание.

Иерархическая структура: Тезаурус APA организует понятия и фразы с использованием иерархической структуры. Понятия более высокого уровня связаны с более точными и узкими терминами, что позволяет осуществлять иерархический просмотр и навигацию по связанным терминам. Эта иерархическая структура помогает пользователям уточнить поиск и изучить концепции в определенных дисциплинах психологии.

Тезаурус APA содержит элементы управления синонимами и перекрестные ссылки для управления различиями в терминологии и обеспечения последовательного поиска важного материала. Синонимы и аналогичные термины снабжены перекрестными ссылками, чтобы пользователи могли перейти к предпочтительным терминам тезауруса. Эта функция повышает точность и эффективность поиска информации, направляя пользователей к наиболее релевантным поисковым фразам [28].

Поддержка нескольких языков: Тезаурус APA преимущественно состоит из английской терминологии, отражающей основной язык, используемый в психологической литературе. Тем не менее, были предприняты усилия по переводу важных терминов на различные языки, чтобы помочь исследователям и специалистам во всем мире.

Тезаурус APA необходим для поиска информации по предмету психологии. Он используется для индексации и поиска литературы по психологии, что позволяет точно и последовательно находить соответствующие

статьи, книги и другие ресурсы. Тезаурус APA помогает исследователям, студентам, клиницистам и профессионалам просматривать обширный психологический материал и находить ресурсы, соответствующие их личным исследовательским интересам или профессиональным потребностям [28].

1.10.3 Тезаурус ERIC

Информационный центр образовательных ресурсов (ERIC) создал и поддерживает тезаурус ERIC, специализированный ограниченный словарь. Он предназначен для облегчения индексации, организации и поиска учебных материалов и ресурсов. Тезаурус ERIC используется ERIC, известной исследовательской базой данных в области образования, для повышения точности и согласованности поиска информации в области образования [29].

Тезаурус ERIC имеет следующие основные особенности:

Тезаурус ERIC охватывает широкий спектр образовательных тем, включая методы обучения, создание учебных программ, педагогическую психологию, оценивание и оценивание, образовательные технологии, специальное образование и другие. Он включает словарный запас и понятия, применимые ко всем уровням образования, от раннего детства до высшей школы.

Стандартизированная терминология: Тезаурус ERIC гарантирует единообразие и стандартизацию терминологии для индексации учебной литературы. Термины тезауруса тщательно отбираются и проверяются специалистами в области образования, чтобы обеспечить их актуальность и точность. Пользователи могут находить конкретные образовательные ресурсы и осуществлять точный поиск благодаря стандартизированной терминологии.

Иерархическая структура: Тезаурус ERIC организует образовательные концепции и терминологию с использованием иерархической структуры. Он организован в виде дерева, где более широкие фразы представляют понятия более высокого уровня, а более узкие термины представляют более частные понятия. Используя эту иерархическую структуру, пользователи могут изучать связанные фразы и углубляться в конкретные области образования, представляющие интерес [29].

Установление отношений: В дополнение к иерархическим связям тезаурус ERIC развивает ассоциативные отношения, отношения эквивалентности и иерархические отношения между понятиями. Эти связи помогают в расширении результатов поиска, предоставлении альтернативных фраз и обнаружении соответствующих образовательных тем.

Многоязычная поддержка: тезаурус ERIC был создан в основном на английском языке, он стремится включать переводы основных терминов на многие языки, чтобы помочь ученым и преподавателям во всем мире. Эта многоязычная возможность повышает доступность и полезность тезауруса для мировой аудитории.

Тезаурус ERIC используется для индексации и поиска учебных материалов в базе данных ERIC. Тезаурус ERIC используется исследователями, преподавателями, политиками и другими заинтересованными сторонами для поиска соответствующих статей, документов и ресурсов по предмету образования. Он помогает пользователям сузить поисковые запросы и найти ресурсы, соответствующие их индивидуальным образовательным требованиям, предлагая стандартизированную терминологию и иерархические связи (24).

Сотрудничество специалистов в области образования, информационных специалистов и команды ERIC необходимо для постоянного создания и поддержки Тезауруса ERIC. Регулярные пересмотры гарантируют, что тезаурус актуален и отражает меняющийся мир образовательных исследований и практики.

1.10.3 Тезаурус AGROVOC

Продовольственная и сельскохозяйственная организация Объединенных Наций (ФАО) создала и поддерживает AGROVOC, многоязычный сельскохозяйственный тезаурус. Он служит полным регулируемым словарем для сельского хозяйства и связанных с ним областей, улучшая организацию, поиск и обмен сельскохозяйственной информацией во всем мире.

Основные характеристики AGROVOC:

AGROVOC предназначен для поддержки различных языков, что позволяет пользователям получать сельскохозяйственную информацию на языке по своему выбору. Он содержит терминологию и концепции на различных языках, включая английский, французский, испанский, арабский, китайский и другие. Многоязычная поддержка повышает доступность и удобство использования AGROVOC для ученых, практиков и политиков из разных мест и языков.

AGROVOC охватывает широкий спектр сельскохозяйственных вопросов, таких как растениеводство, животноводство, агролесоводство, экономика сельского хозяйства, продовольственная безопасность, развитие сельских районов и многое другое. Он включает сельскохозяйственную лексику и такие темы, как методы ведения сельского хозяйства, сельскохозяйственные технологии, сельскохозяйственная политика и устойчивое сельское хозяйство.

Стандартизированная терминология: AGROVOC гарантирует, что терминология управления сельскохозяйственной информацией стандартизирована и непротиворечива. Тезаурус содержит термины, которые были тщательно отобраны и утверждены для описания определенных сельскохозяйственных идей. Использование стандартизированной номенклатуры улучшает совместимость и интеграцию сельскохозяйственных данных, а также способствует эффективному поиску информации и обмену ею.

Иерархическая структура: AGROVOC организует сельскохозяйственные понятия и слова, используя иерархическую структуру. Он организован в виде иерархической древовидной структуры, где более широкие термины (понятия

более высокого уровня) связаны с более узкими фразами (понятиями более низкого уровня). Эта иерархическая структура позволяет посетителям перемещаться между связанными темами и изучать конкретные сельскохозяйственные области [30].

AGROVOC использует междисциплинарный подход, охватывающий несколько дисциплин в сельском хозяйстве и смежных отраслях. Он охватывает такие области, как агрономия и почвоведение, а также сельскохозяйственная инженерия, сельская социология и сельскохозяйственная политика. Многопрофильный охват AGROVOC гарантирует, что он отвечает уникальным потребностям клиентов во многих сельскохозяйственных секторах и исследовательских дисциплинах.

AGROVOC играет решающую роль в поддержке управления сельскохозяйственной информацией, интеграции данных и обмена знаниями. Исследователи, практики, политики и специалисты в области информации широко используют его для индексирования и извлечения сельскохозяйственных ресурсов, таких как научные публикации, отчеты, статистические данные и материалы по распространению сельскохозяйственных знаний. Стандартизированная номенклатура и многоязычная поддержка, предоставляемая AGROVOC, помогают быстро находить и извлекать материалы, способствуя эффективному сотрудничеству и обмену знаниями в мировом сельскохозяйственном сообществе [31].

Сельскохозяйственные эксперты, терминологи и официальные лица ФАО сотрудничают в постоянном создании и поддержании AGROVOC. Регулярные обновления и переговоры с профессионалами в данной области гарантируют, что AGROVOC остается актуальным, отражает развивающиеся тенденции в сельском хозяйстве и удовлетворяет растущие информационные потребности сельскохозяйственного сообщества.

2.Анализ

2.1 Анализ исследования

Разработка тезауруса: В ходе исследования был успешно создан тезаурус с использованием автоматизированных методов обработки текстов в заданной предметной области с акцентом на финансово-экономической сфере бизнеса. Тезаурус представлял собой тщательно подобранный список слов и фраз, выбранных наугад из огромного объема текстовых материалов. Ручное редактирование и уточнение фраз привели к тому, что окончательный тезаурус содержал меньше терминов, особенно часто встречающихся и имеющих синонимы.

Состав тезауруса: Состав тезауруса раскрывает интригующие тенденции в длине терминов. Большинство терминов (76% от общего числа) были терминами, состоящими из одного слова. Термины из двух слов составляют 21% тезауруса, а термины из трех и более слов составляют оставшиеся 3%.

2.2 Объяснение выводов

Этот раздел предоставляет возможность обсудить и интерпретировать результаты и результаты в связи с целями исследования по созданию программного приложения с использованием методов семантической обработки и разработки тезауруса для поиска информации. Это позволяет более глубоко изучить последствия, ограничения и значимость результатов исследования.

Последствия семантического поиска на основе тезауруса: Обсуждение углубляется в последствия использования методов семантического поиска на основе тезауруса в разработанном программном приложении. В нем исследуется, как интеграция семантических отношений и синонимов повышает точность и актуальность поиска информации, что приводит к более точным и значимым результатам. Подчеркнуты преимущества улучшенных возможностей поиска и расширенного анализа данных для пользователей и организаций.

Практическое применение и преимущества. В ходе обсуждения рассматриваются практические применения и преимущества программного приложения и тезауруса. Он исследует, как приложение может использоваться в реальных условиях, таких как финансы и экономика, чтобы обеспечить более эффективный и действенный поиск информации. Обсуждение сосредоточено на том, как программное обеспечение помогает лучшему принятию решений, исследованиям и получению знаний в соответствующей теме.

В докладе признаются ограничения и препятствия, с которыми пришлось столкнуться при создании и развертывании программного обеспечения и тезауруса. В нем рассматриваются такие вопросы, как объем и покрытие тезауруса, возможность отсутствия или неполных семантических связей, а также

необходимость постоянного обновления и обслуживания. Ограничения и проблемы открывают возможности для будущих исследований и достижений в семантической обработке и построении тезауруса.

Сравнение с существующими методологиями: В докладе сравнивается предлагаемая программа и тезаурус с существующими методологиями и инструментами поиска информации. Он оценивает сильные и слабые стороны предлагаемого решения по сравнению с другими семантическими поисковыми системами, тезаурусами или базами данных. Обсуждение сосредоточено на отличительных особенностях и преимуществах созданного приложения, а также на возможностях будущих улучшений и сотрудничества.

2.3 Требования к базе данных

Проектирование базы данных — это процесс создания проекта базы данных для поддержки операций бизнес-объекта и достижения его целей. Это трудоемкий процесс, который требует совместных усилий аналитиков, дизайнеров и пользователей. При проектировании базы данных имейте в виду, что база данных должна соответствовать ряду требований. Этими требованиями являются: 1) целостность базы данных - требование полноты и непротиворечивости данных; 2) повторное использование данных; 3) оперативный поиск и получение информации по запросу пользователей; 4) Легко обновлять данные: 5) Уменьшать чрезмерную избыточность данных; 6) Защита данных от несанкционированного доступа, фальсификации и уничтожения

2.3.1 БД SQLite

QLite — это быстрая и легкая встраиваемая однофайловая СУБД на языке C, которая не имеет сервера и позволяет хранить всю базу локально на одном устройстве. Для работы SQLite не нужны сторонние библиотеки или службы.

Насколько надежна БД SQLite?

При выпуске версии она проходит через ряд серьезнейших автоматических тестов , покрытие кода тестами 100%.

Какие инструменты доступны разработчикам ?

Доступна консольная утилита для работы с базами (sqlite3.exe, «a command-line shell for accessing and modifying SQLite databases»). Понятие «встраиваемый» означает, что СУБД не использует парадигму клиент-сервер [32]. Движок SQLite —не отдельно работающий процесс, с которым взаимодействует программа, а библиотека. Программа компонуется с ней, и движок служит составной частью программы. В качестве протокола обмена применяются вызовы функций (API) библиотеки SQLite.

Благодаря свойствам SQLite применяется:

- на сайтах с низким и средним трафиком;
- в локальных однопользовательских, мобильных приложениях или играх, не предназначенных для масштабирования;
- в программах, которые часто выполняют прямые операции чтения/записи на диск;
- в приложениях для тестирования бизнес-логики.

2.4 Предполагаемый результат

Предлагаемый тезаурус имеет практическое значение для поиска информации, особенно в финансовой и экономической областях. Это отличный ресурс для расширения возможностей поиска, улучшения категоризации документов и повышения точности процедур поиска информации.

Вклад в знания: исследование дополняет совокупность знаний в области построения тезауруса и семантической обработки. В нем подчеркивается эффективность автоматизированных подходов к созданию тезаурусов и дается представление о разработке, уточнении и практических последствиях созданного тезауруса.

В целом выводы и результаты этой работы свидетельствуют об успешном создании тезауруса с использованием автоматизированных подходов, дают представление о его составе и подчеркивают его практическое использование при поиске информации. Статья дополняет имеющиеся знания и понимание процессов разработки тезаурусов и их важности для улучшения организации и поиска информации в конкретных дисциплинах.

3.Практическая часть

3.1 Создание базы данных

Для более удобного и универсального хранения данных, будет использоваться БД SQLite. Для неё есть несколько удобных библиотек, с которыми достаточно просто работать. Запросы к БД будут написаны на SQL. Будут реализованы методы доступа к БД, чтобы можно было работать через управляющего посредника. База данных будет хранится рядом с основным файлом программы.

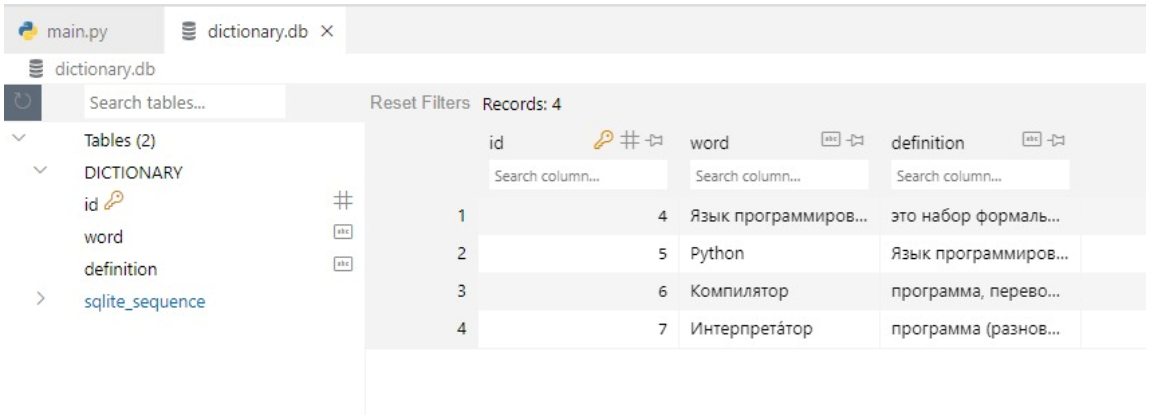


Рисунок 2.1 База данных

Хранение данных на платформе БД SQLite(рисунок 2.1). В этом окне сохраняется база данных, чтобы предотвратить потери.

```
145 class DAO:
146     def __init__(self):
147         self.connection = ''
148         self.add_to_base_sql = 'INSERT INTO DICTIONARY (word, definition) values (?, ?)'
149         self.update_sql = 'UPDATE DICTIONARY SET word = ?, definition = ? where id = ?;'
150         self.tmp_word=''
151         self.select_from_db_by_word_sql = "SELECT * FROM DICTIONARY WHERE word LIKE ? ORDER by word"
152         self.select_all_from_db = 'SELECT * FROM DICTIONARY ORDER by word'
153         self.delete_from_db_by_id_sql = 'DELETE FROM DICTIONARY WHERE id = ?'
154         self.file_name = "dictionary.db"
155 
156         if os.path.isfile("dictionary.db"):
157             self.connection = sl.connect('dictionary.db')
158         else:
159             self.connection = sl.connect('dictionary.db')
160             with self.connection:
161                 self.connection.execute(
162                     """
163                         CREATE TABLE DICTIONARY (
164                             id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
165                             word TEXT,
166                             definition TEXT
167                         );
168                     """
169                 )
170 
```

Рисунок 2.2 Создание базы данных

Код является частью класса, который управляет словарем с помощью базы данных SQLite(рисунок 2.2). Давайте пройдемся по коду шаг за шагом:

Сначала он проверяет, существует ли файл «dictionary.db» в текущем каталоге, используя функцию os.path.isfile().

Если файл существует, он подключается к существующей базе данных с помощью sl.connect('dictionary.db') и присваивает объект соединения self.connection.

Если файл не существует, он создает новый файл базы данных, подключаясь к новой базе данных SQLite с помощью sl.connect('dictionary.db') и присваивая объект соединения self.connection.

Если файл не существовал и была создана новая база данных, код переходит к созданию новой таблицы с именем "DICTIONARY" в базе данных.

В таблице есть следующие столбцы:

"id" (INTEGER PRIMARY KEY AUTOINCREMENTавтоматически генерируемый уникальный идентификатор для каждой записи.

«lang» (TEXT DEFAULT "Python"): язык слова со значением по умолчанию «Python».

"слово" (TEXT): определяемое слово.

"определение" (TEXT): Определение слова.

"kk_title" (TEXT DEFAULT ""): название слова на другом языке со значением по умолчанию пустой строки.

"kk_definition" (TEXT DEFAULT ""): определение слова на другом языке со значением по умолчанию пустой строки.

«владелец» (TEXT DEFAULT "Base")владелец слова со значением по умолчанию «Base».

После создания таблицы код продолжает вставлять данные в таблицу «DICTIONARY», перебирая db_init_data, который, вероятно, представляет собой список кортежей, представляющих пары слов и определений.

Для каждого кортежа в db_init_data код выполняет оператор SQL для вставки слова и определения в таблицу с помощью метода self.connection.execute().

В целом, этот код проверяет, существует ли файл базы данных, создает новый файл базы данных, если он не существует, создает таблицу с именем "DICTIONARY", если это была новая база данных, и вставляет исходные данные в таблицу. Цель состоит в том, чтобы настроить и инициализировать базу данных словаря для дальнейших операций.

```
171     def add(self, word, definition):
172         data = (word, definition)
173         cur =  self.connection.cursor()
174         cur.execute(self.add_to_base_sql, data)
175         self.connection.commit()
```

Рисунок 2.3 Создание панели управления

Код представляет собой набор методов внутри одного класса, которые взаимодействуют с базой данных словаря.

get_langs(self): этот метод извлекает все отдельные языки (языки), хранящиеся в таблице DICTIONARY. Он выполняет следующие шаги:

Он создает объект курсора (cur), используя соединение.

Выполняет оператор SQL для выбора различных языков из таблицы DICTIONARY.

Извлекает все результаты с помощью cur.fetchall(), которая возвращает список кортежей.

Перебирает ответ и извлекает язык из каждого кортежа, преобразует его в строку и добавляет в список ответов.

Наконец, он возвращает список языков.

add(self, lang, word, definition, owner): этот метод добавляет новую запись в таблицу DICTIONARY. Он принимает параметры lang (language), self, word, definition, owner в качестве входных данных.

Необходимые шаги:

Создание кортежа с именем data, содержащий значения для новой записи.

Создание объект курсора (cur), используя соединение.

Выполнение оператора SQL self.add_to_base_sql для вставки данных в таблицу DICTIONARY.

Фиксирует изменения в базе данных с помощью self.connection.commit().

update(self, id, lang, word, определение, kk_title, kk_definition): этот метод обновляет существующую запись в таблице DICTIONARY на основе заданного идентификатора. Параметры lang, word, определение, kk_title и kk_definition представляют обновленные значения (рисунок 2.3).

```
177     def update(self, id, word, definition):
178         data = (word, definition, id)
179         cur =  self.connection.cursor()
180         cur.execute(self.update_sql, data)
181         self.connection.commit()
```

Рисунок 2.4 Создание панели управления

Код выполняет следующие шаги:

Создание кортежа с именем data, содержащий обновленные значения и идентификатор обновляемой записи.

Создание объекта курсора (cur), используя соединение.

Выполнение оператора SQL self.update_sql для обновления соответствующей строки в таблице DICTIONARY.

Фиксирует изменения в базе данных.

remove(self, id): этот метод удаляет запись из таблицы DICTIONARY на основе заданного идентификатора(рисунок 2.4).

```
182
183     def remove(self, id):
184         data = (id,)
185         cur =  self.connection.cursor()
186         cur.execute(self.delete_from_db_by_id_sql, data)
187         self.connection.commit()
188
```

Рисунок 2.5 Создание панели управления

Код выполняет следующие шаги:

Создание кортежа с именем data, содержащий идентификатор удаляемой записи.

Создание объект курсора (cur), используя соединение.

Выполняет оператор SQL self.delete_from_db_by_id_sql для удаления соответствующей строки из таблицы DICTIONARY.

Фиксирует изменения в базе данных.

get_all(self): этот метод извлекает все записи из таблицы DICTIONARY (рисунок 2.5).

```
189     def get_all(self):
190         cur =  self.connection.cursor()
191         cur.execute(self.select_all_from_db)
192         responce = cur.fetchall()
193         return responce
```

Рисунок 2.6 Создание панели управления

Код выполняет следующие шаги:

Создает объект курсора (cur), используя соединение.

Выполняет оператор SQL self.select_all_from_db для выбора всех строк из таблицы DICTIONARY.

Извлекает все результаты с помощью cur.fetchall(), которая возвращает список кортежей, представляющих записи.

get_by_word(self, word): этот метод извлекает записи из таблицы DICTIONARY, которые соответствуют заданному слову (частично или полностью) (рисунок 2.6).

```
194
195     def get_by_word(self, word):
196         cur =  self.connection.cursor()
197         cur.execute(self.select_from_db_by_word_sql, ("%" + word + "%",))
198         responce = cur.fetchall()
199         return responce
```

Рисунок 2.7 Создание панели управления

Код выполняет следующие шаги:

Создает объект курсора (cur), используя соединение.

Выполняет оператор SQL, self.select_from_db_by_word_sql, для выбора строк из таблицы DICTIONARY, в которых есть данное слово в столбцах word, определения или kk_definition (используя подстановочный знак % для частичных совпадений).

Извлекает все результаты с помощью cur.fetchall(), которая возвращает список кортежей, представляющих записи (рисунок 2.7).

3.2 Создание интерфейса

Python имеет множество фреймворков с графическим интерфейсом, но Tkinter — единственный фреймворк, встроенный в стандартную библиотеку Python. У Tkinter есть несколько сильных сторон. Он кроссплатформенный, поэтому один и тот же код работает в Windows, macOS и Linux. Визуальные элементы визуализируются с использованием собственных элементов операционной системы, поэтому приложения, созданные с помощью Tkinter, выглядят так, как будто они принадлежат платформе, на которой они выполняются. Изначально была создана первая версия интерфейса, которая показывает функционал программы (рисунок 3.1,3.2).

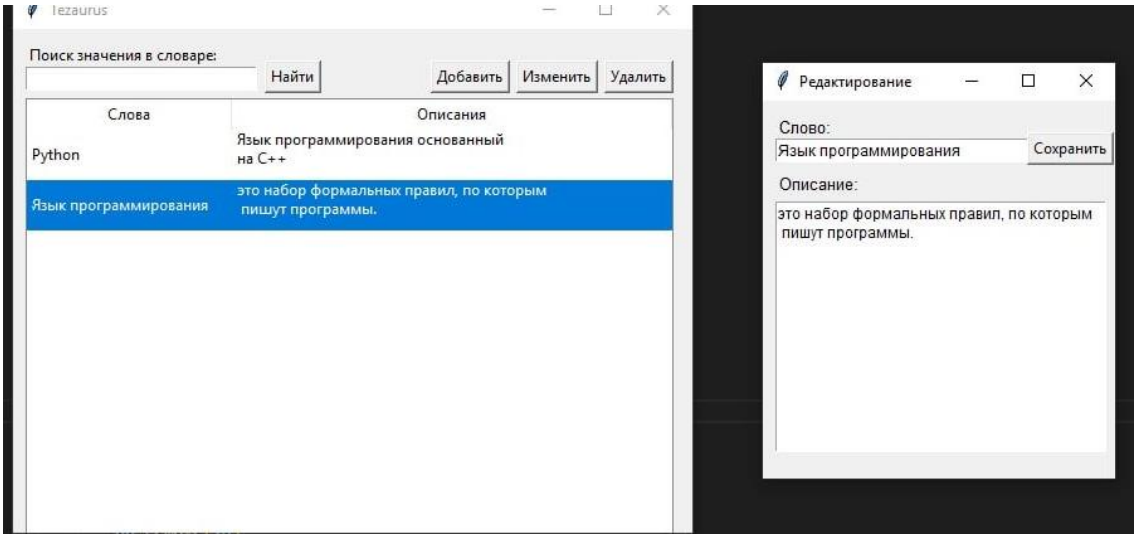


Рисунок 3.1 Первая версия интерфейса

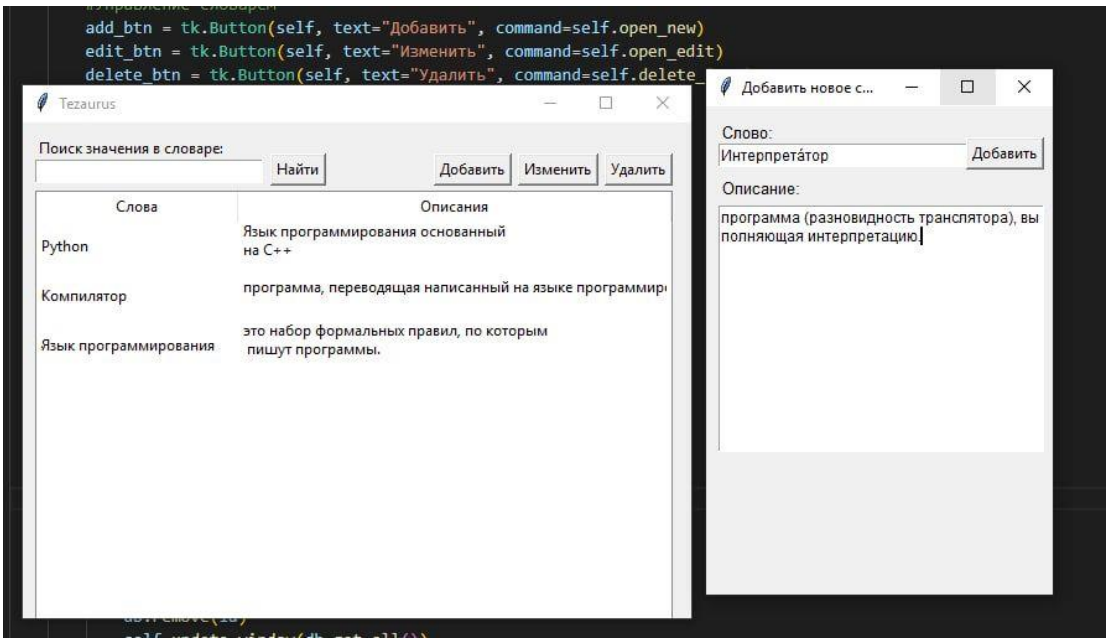


Рисунок 3.2 Первая версия интерфейса

3.3 Описание программы

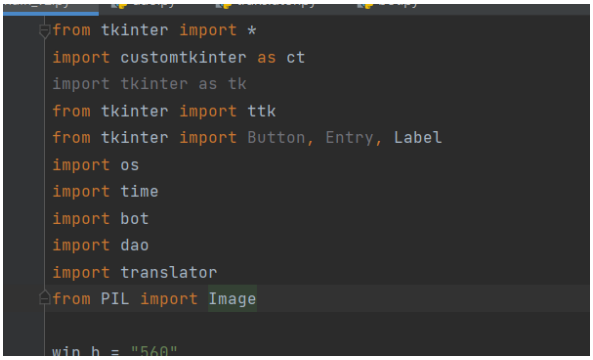


Рисунок 4.1 Основной файл программы

Код представляет собой сценарий, который импортирует различные модули и библиотеки и настраивает графический интерфейс пользователя (GUI) с использованием библиотеки Tkinter.

from tkinter import *: импортирует все классы, функции и переменные из модуля Tkinter, что позволяет использовать их без префикса модуля.

import customtkinter as ct: импортирует модуль с именем customtkinter и назначает ему псевдоним ct. Этот модуль, вероятно, содержит пользовательские классы или функции для элементов графического интерфейса Tkinter.

import tkinter as tk: импортирует модуль Tkinter и назначает ему псевдоним tk. Это позволяет вам получить доступ к классам и функциям Tkinter, используя префикс tk.

from tkinter import ttk: импортирует дополнительные классы Tkinter для тематических виджетов. ttk означает «тематические виджеты Tkinter».

from tkinter import Button, Entry, Label: импортирует определенные классы (Button, Entry, Label) из модуля Tkinter.

import os: импортирует модуль os, который обеспечивает способ взаимодействия с операционной системой, например доступ к файлам и каталогам.

Import time: импортирует модуль времени, который предоставляет функции для работы с задачами, связанными со временем, такими как ожидание в течение указанной продолжительности.

from PIL import Image: импортирует модуль Image из библиотеки PIL (Python Imaging Library). PIL предоставляет возможности обработки изображений (рисунок 4.1).

Основной файл программы - main.py, в котором описана логика работы всей программы, а также подключаемых модулей. В самом начале файла объявляется набор необходимых библиотек, для работы программы. Первые три строки отвечают за подключение библиотеки TKinter. Она необходима для создания графического интерфейса приложения.

Библиотека Time - втроенная в Python библиотека. Она упрощает всю работу со временем.

Следующие три модуля - созданные мною в рамках программы модули. Bot - работа с нейросетью, DAO - работа с базой данных, Translator - работа с переводом текста.

Для того, чтобы понять всю структуру и логику работы программы, разберем самописные модули. Модуль Bot базируется на библиотеке openai, которая находится в открытом доступе.

Данная библиотека позволяет реализовать простую работу с различными нейронными сетями и обученными моделями. Для работы необходимо произвести регистрацию на сайте openai и получить свой api ключ.

```python
import time
import openai


openai.api_key = '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'
engine = 'text-davinci-003'

def ask(prompt, lang=''):
    prompt = prompt + ' ' + lang
    completion = openai.Completion.create(engine=engine,
                                          prompt=prompt,
                                          temperature=0.5,
                                          max_tokens=700)
    return completion.choices[0]['text'][:]

```

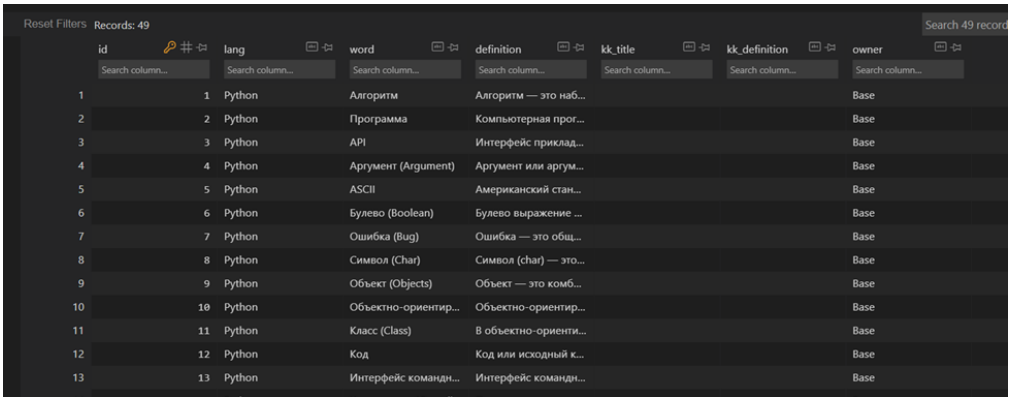
Рисунок 4.2 Использование модели Давинчи.

Далее заранее выбрав экспериментальным путем обученную модель нейронной сети, а в нашем случае это text-davinci-003, заносим это значение в переменную engine для дальнейшего использования. Затем реализовав функцию ask, мы можем делать запросы нейронной сети. ASK принимает в себя два параметра, prompt и lang (рисунок 4.2). Сделано это для универсальности использования программы. prompt - текст запроса для бота. Внутри функции вызывается метод create, который возвращает массив различных объектов. Из него мы и получаем наш ответ(рисунок 4.3).

```python
    def get_langs(self):
        cur =  self.connection.cursor()
        cur.execute("SELECT DISTINCT lang FROM DICTIONARY")
        responce = cur.fetchall()
        return responce

    def add(self, lang, word, definition, owner):
        data = (lang, word, definition, owner)
        cur =  self.connection.cursor()
        cur.execute(self.add_to_base_sql, data)
        self.connection.commit()

    def update(self, id, lang, word, definition, kk_title, kk_definition):
        data = (word, definition, lang, kk_title, kk_definition,id)
        cur =  self.connection.cursor()
        cur.execute(self.update_sql, data)
        self.connection.commit()

    def remove(self, id):
        data = (id,)
        cur =  self.connection.cursor()
        cur.execute(self.delete_from_db_by_id_sql, data)
        self.connection.commit()

    def get_all(self):
        cur =  self.connection.cursor()
        cur.execute(self.select_all_from_db)
        responce = cur.fetchall()
        return responce

    def get_by_word(self, word):
        cur =  self.connection.cursor()
        cur.execute(self.select_from_db_by_word_sql, ("%" + word + "%", "%" + word + "%", "%" + word + "%",))
        responce = cur.fetchall()
        return responce

```

Рисунок 4.3 Использование модели Давинчи.

Модуль DAO отвечает за работу с базой данных. Весь модуль построен вокруг библиотеки sqlite3 и os. sqlite3 позволяет работать с sql базами данных, а библиотека os со средствами операционной системы. В нашем конкретном случае она будет использоваться для работы с файлом базы данных dictionary.db. Также в случае удаления файла базы данных, она будет создана вновь из набора заложенных терминов. Реализован в модуле стандартный набор CRUD операций (create, read, update, delete).



Reset Filters Records: 49 Search 49 record

	id	lang	word	definition	kk_title	kk_definition	owner
	Search column...	Search column...	Search column...	Search column...	Search column...	Search column...	Search column...
1	1	Python	Алгоритм	Алгоритм — это наб...			Base
2	2	Python	Программа	Компьютерная прог...			Base
3	3	Python	API	Интерфейс приклад...			Base
4	4	Python	Аргумент (Argument)	Аргумент или аргум...			Base
5	5	Python	ASCII	Американский стан...			Base
6	6	Python	Булево (Boolean)	Булево выражение ...			Base
7	7	Python	Ошибка (Bug)	Ошибка — это общ...			Base
8	8	Python	Символ (Char)	Символ (char) — это...			Base
9	9	Python	Объект (Objects)	Объект — это комб...			Base
10	10	Python	Объектно-ориентир...	Объектно-ориентир...			Base
11	11	Python	Класс (Class)	В объектно-ориенти...			Base
12	12	Python	Код	Код или исходный к...			Base
13	13	Python	Интерфейс командн...	Интерфейс командн...			Base

Рисунок 4.4 База данных.

В базе данных используются несколько полей для идентификации термина:

Id-уникальный номер термина для базы

Lang – категория термина. Обычно используется язык программирования

Word – термин

Definition – описание

Kk_title – термин на казахском языке

Kk_definition – описание на казахском языке

Owner – тот, кто добавил термин в базу. Bot – запрос в нейронную сеть, user – человек, base – начальный термин(рисунок 4.4).

```python
from deep_translator import GoogleTranslator

def translate(text):
    translated = GoogleTranslator(source="ru", target="kk").translate(text=text)
    return translated
```

Рисунок 4.5 Модуль Translator.

Модуль содержит всего лишь одну функцию – translate. В параметрах функции уже заложен базовый язык и конечный язык: source и target. В качестве возвращаемого значения, функция передает переведенное слово и описание.

Фрагмент кода демонстрирует функцию translate(), которая использует библиотеку deep_translator для выполнения языкового перевода с помощью Google Translate(рисунок 4.5).

Код импортирует класс GoogleTranslator из модуля deep_translator.

Функция translate() принимает текстовый параметр, представляющий текст, который необходимо перевести.

Внутри функции создается экземпляр класса GoogleTranslator с указанием исходного языка как «ru» (русский) и целевого языка как «kk» (казахский).

Вызывается метод translate() экземпляра GoogleTranslator, который передает текстовый параметр для перевода. Этот метод выполняет перевод с использованием указанных исходного и целевого языков.

Переведенный текст присваивается переведенной переменной.

Наконец, переведенный текст возвращается из функции.

Чтобы использовать эту функцию, необходимо установить библиотеку deep_translator. Он предоставляет удобный способ перевода текста с помощью различных сервисов перевода. В этом случае он использует Google Translate в качестве поставщика перевода.

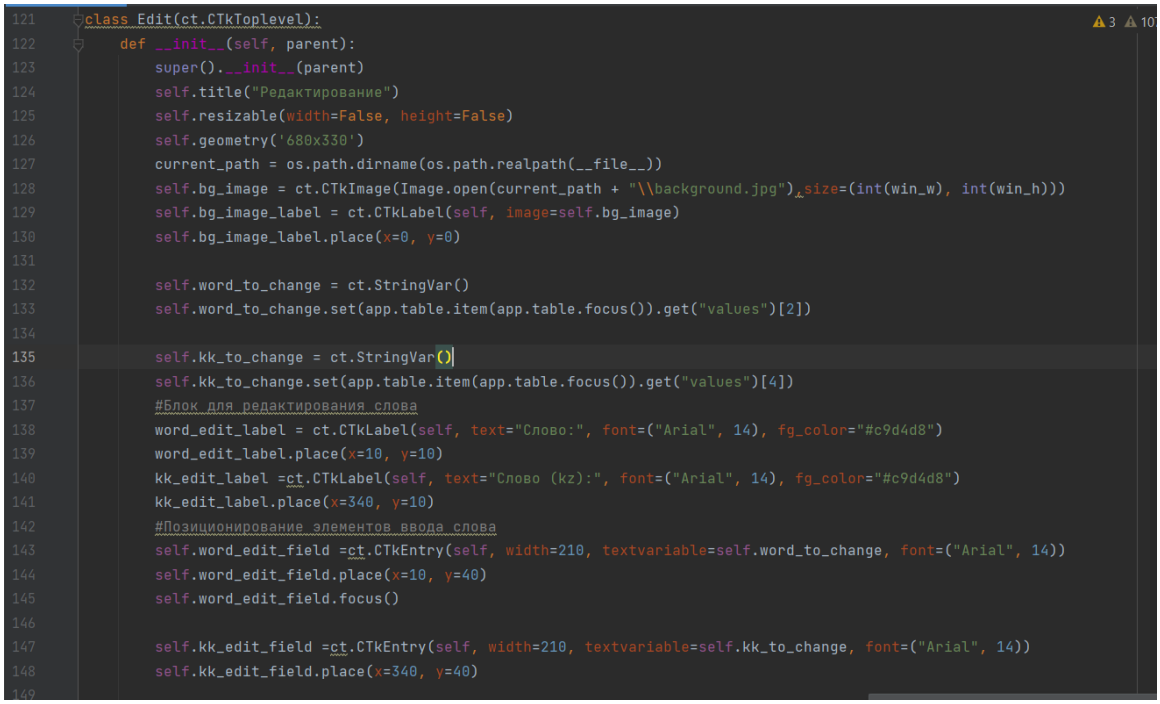
```
360  if __name__ == "__main__":
361      db = dao.DAO()
362      app = App()
363      app.resizable(width=False, height=False)
364      app.mainloop()
365
```

Рисунок 4.6 Файл main.py.

```
23   #Класс главного окна, в котором отображается все данные: словарь, кнопки
     управления и т.д.
24 > class App(tk.Tk): ···
125
126  #Класс для работы с окном редактирования
127 > class Edit(tk.Toplevel): ···
209
210  #Класс окна для добавления нового элемента в словарь
211 > class New(tk.Toplevel): ···
271
272  #Модуль импорта нескольких терминов
273 > class Term_table(tk.Toplevel): ···
341
342  #Окно информации о студенте
343 > class Info(tk.Toplevel): ···
359
```

Рисунок 4.7 Классы окон.

Каждый из этих классов отвечает за создание окон: главное окно программы, окно редактирования и перевода, окно добавления нового термина, окно добавления нескольких терминов и окно справки (порядок соответствует коду) (рисунок 4.7).



```
class Edit(ct.CTkToplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.title("Редактирование")
        self.resizable(width=False, height=False)
        self.geometry('680x330')
        current_path = os.path.dirname(os.path.realpath(__file__))
        self.bg_image = ct.CTkImage(Image.open(current_path + "\\background.jpg"), size=(int(win_w), int(win_h)))
        self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
        self.bg_image_label.place(x=0, y=0)

        self.word_to_change = ct.StringVar()
        self.word_to_change.set(app.table.item(app.table.focus()).get("values")[2])

        self.kk_to_change = ct.StringVar()
        self.kk_to_change.set(app.table.item(app.table.focus()).get("values")[4])
        #Блок для редактирования слова
        word_edit_label = ct.CTkLabel(self, text="Слово:", font=("Arial", 14), fg_color="#c9d4d8")
        word_edit_label.place(x=10, y=10)
        kk_edit_label =ct.CTkLabel(self, text="Слово (kz):", font=("Arial", 14), fg_color="#c9d4d8")
        kk_edit_label.place(x=340, y=10)
        #Позиционирование элементов ввода слова
        self.word_edit_field =ct.CTkEntry(self, width=210, textvariable=self.word_to_change, font=("Arial", 14))
        self.word_edit_field.place(x=10, y=40)
        self.word_edit_field.focus()

        self.kk_edit_field =ct.CTkEntry(self, width=210, textvariable=self.kk_to_change, font=("Arial", 14))
        self.kk_edit_field.place(x=340, y=40)
```

Рисунок 4.8.1 Класс окна Edit.

Фрагмент кода определяет класс с именем Edit, который наследуется от ct.CTkToplevel. Похоже, это пользовательская реализация окна Tkinter Toplevel для редактирования записей словаря.

Метод __init__() является конструктором класса Edit. Он инициализирует окно верхнего уровня, вызывая конструктор родительского класса (ct.CTkToplevel), задает для заголовка окна значение «Edit» (Edit) и настраивает размер и геометрию окна.

Атрибуты bg_image и bg_image_label используются для установки фонового изображения для окна. Изображение загружается из файла с именем «background.jpg» с помощью функции Image.open() из библиотеки PIL. Классы ct.CTkImage и ct.CTkLabel из модуля customtkinter используются для отображения фонового изображения.

Несколько переменных StringVar (word_to_change, kk_to_change, lang_to_change) создаются для хранения значений выбранной редактируемой словарного листа.

Код создает метки, поля ввода и текстовые поля для редактирования слова, его казахского перевода, языка и определения. Для этих элементов графического интерфейса используются классы ct.CTkLabel, ct.CTkEntry и ct.CTkTextbox из модуля customtkinter.

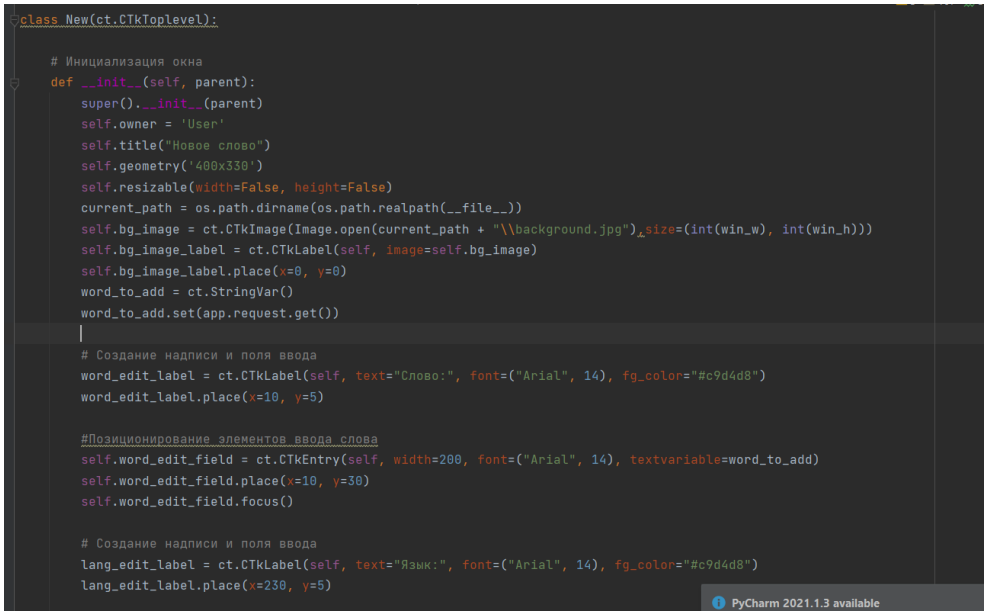
Поле lang_edit_field представляет собой поле со списком (выпадающее меню), заполненное параметрами языка, полученными из функции db.get_langs(). Позволяет выбрать язык редактируемого слова.

Текстовые поля word_definition и kk_word_definition предварительно заполнены существующими определениями выбранного слова из таблицы.

Кнопки сохранения изменений (save_btn), удаления записи (delete_btn), отмены редактирования (cancel_btn) и перевода слова на казахский язык (translate_btn) создаются с помощью класса ct.CTkButton.

На методы update_record(), delete() и translate_kk() ссылаются как на соответствующие команды для кнопок.

Код зависит от модуля customtkinter, который не является частью стандартной библиотеки Tkinter. Модуль customtkinter, вероятно, содержит пользовательские классы и функции для элементов графического интерфейса с дополнительным стилем или функциональностью. Чтобы полностью понять код и успешно его запустить, потребуется доступ к модулю customtkinter и его зависимостям (рисунок 4.8.1).



```
class New(ct.CTkToplevel):

    # Инициализация окна
    def __init__(self, parent):
        super().__init__(parent)
        self.owner = 'User'
        self.title("Новое слово")
        self.geometry('400x330')
        self.resizable(width=False, height=False)
        current_path = os.path.dirname(os.path.realpath(__file__))
        self.bg_image = ct.CTkImage(Image.open(current_path + "\\background.jpg"), size=(int(win_w), int(win_h)))
        self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
        self.bg_image_label.place(x=0, y=0)
        word_to_add = ct.StringVar()
        word_to_add.set(app.request.get())

        # Создание надписи и поля ввода
        word_edit_label = ct.CTkLabel(self, text="Слово:", font=("Arial", 14), fg_color="#c9d4d8")
        word_edit_label.place(x=10, y=5)

        #Позиционирование элементов ввода слова
        self.word_edit_field = ct.CTkEntry(self, width=200, font=("Arial", 14), textvariable=word_to_add)
        self.word_edit_field.place(x=10, y=30)
        self.word_edit_field.focus()

        # Создание надписи и поля ввода
        lang_edit_label = ct.CTkLabel(self, text="Язык:", font=("Arial", 14), fg_color="#c9d4d8")
        lang_edit_label.place(x=230, y=5)
```

PyCharm 2021.1.3 available

Рисунок4.8.2 Класс окна New.

Фрагмент кода определяет класс с именем New, который наследуется от ct.CTkToplevel. Это пользовательская реализация окна Tkinter Toplevel для добавления новых записей словаря. Вот разбивка кода:

Метод __init__() является конструктором класса New. Он инициализирует окно верхнего уровня, вызывая конструктор родительского класса (ct.CTkToplevel), устанавливает заголовок окна на «Новое слово» (New Word) и настраивает размер и геометрию окна.

Код устанавливает фоновое изображение для окна с помощью классов ct.CTkImage и ct.CTkLabel из модуля customtkinter.

Переменная StringVar с именем word_to_add создается и инициализируется значением, полученным из app.request.get().

Кнопки выполнения сетевого запроса (search_btn), сохранения новой записи в базе данных (save_btn) и отмены операции (cancel_btn) создаются с помощью класса ct.CTkButton. Соответствующие команды указаны как self.ask_network(), self.add_to_db() и self.destroy().

Метод add_to_db() вызывается при нажатии кнопки «Сохранить» (Сохранить). Он извлекает значения из элементов графического интерфейса и использует метод db.add() для добавления новой записи в базу данных. Затем он обновляет окно приложения, вызывая app.update_window(), и удаляет текущее окно.

Метод ask_network() вызывается при нажатии кнопки "Запрос в сети". Он устанавливает для атрибута owner значение «bot» и использует функцию bot.ask() для отправки сетевого запроса с введенным словом. Ответ отображается в текстовом поле word_definition.

Этот код зависит от модуля customtkinter и некоторых внешних переменных (win_w, win_h, app, db, bot) (рисунок 4.8.2).

```
class Term_table(ct.CTkToplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.owner = 'User'
        self.title("Поиск множества")
        self.resizable(width=False, height=False)
        self.geometry('640x410')
        current_path = os.path.dirname(os.path.realpath(__file__))
        self.bg_image = ct.CTkImage(Image.open(current_path + "\\background.jpg"),size=(int(win_w), int(win_h)))
        self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
        self.bg_image_label.place(x=0, y=0)

        #Блок для редактирования слова
        term_edit_label =ct.CTkLabel(self, text="Объект поиска:", font=("Arial", 14), fg_color="#c9d4d8")
        term_edit_label.place(x=10, y=10)

        #Позиционирование элементов ввода слова
        self.term_edit_field =ct.CTkEntry(self, width=210, font=("Arial", 14))
        self.term_edit_field.place(x=10, y=40)
        self.term_edit_field.focus()

        #Панель описания
        lang_label =ct.CTkLabel(self, text="Язык программирования:", font=("Arial", 14), fg_color="#c9d4d8")
        lang_label.place(x=250, y=10)

        self.time_var = ct.StringVar()
        self.time_var.set("")
        self.time_label =ct.CTkLabel(self, textvariable=self.time_var, font=("Arial", 14), fg_color="#c9d4d8")
        self.time_label.place(x=10, y=390)
```

Рисунок 4.8.3 Класс окна Term_table .

Предоставленный вами фрагмент кода определяет класс с именем Term_table, который наследуется от ct.CTkToplevel. Похоже, это пользовательская реализация окна Tkinter Toplevel для поиска и добавления нескольких терминов в словарь.

Метки и поля ввода созданы для ввода условия поиска и выбора языка программирования. Для этих элементов графического интерфейса используются классы ct.CTkLabel и ct.CTkEntry из модуля customtkinter.

Поле со списком с именем lang_edit_field создается и заполняется параметрами языка, полученными из функции db.get_langs(). Позволяет выбрать язык программирования для поиска или добавленных терминов.

Метка и текстовое поле создаются для отображения результатов поиска или добавления определений терминов. Для этих элементов графического интерфейса используются классы ct.CTkLabel и ct.CTkTextbox из модуля customtkinter. Инструкции по использованию окна хранятся в переменной info_text(рисунок 4.8.3).

3.4 Финальная версия интерфейса

Тезаурус выполняет функцию хранение данных в базе, добавление, удаление и редактирование слов/описания. Так же существует окно, где производится запрос в сеть на нахождение новых терминов и добавление их в общую базу данных. При каждым выходе из программы все данные сохраняются в базе данных. Производится перевод термина и описания на казахский язык при помощи функции translator (рисунок 5.1-5.4).

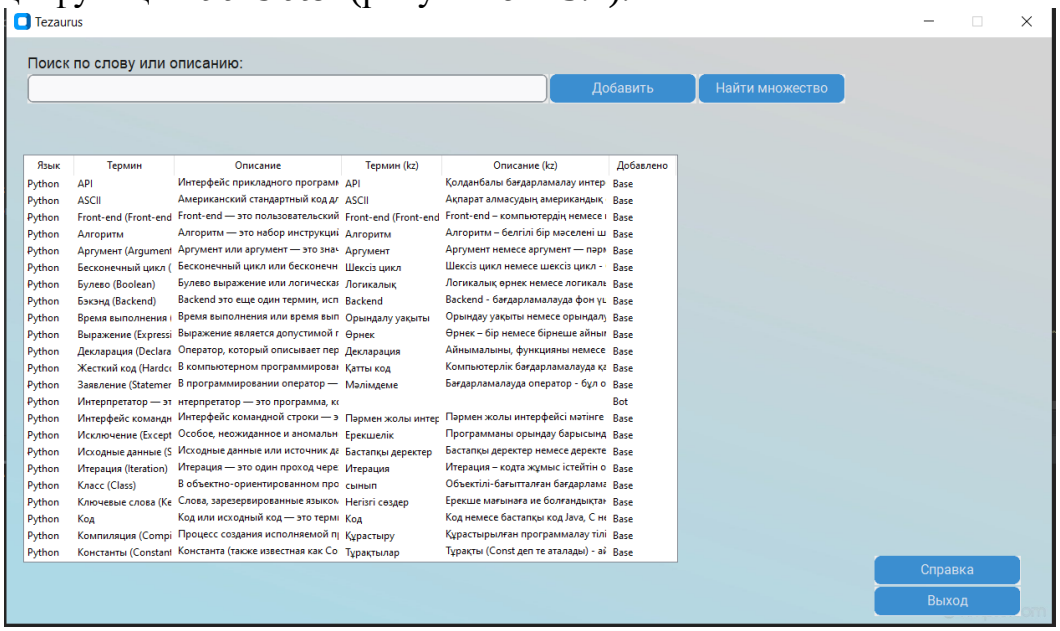


Рисунок 5.1 Интерфейс.

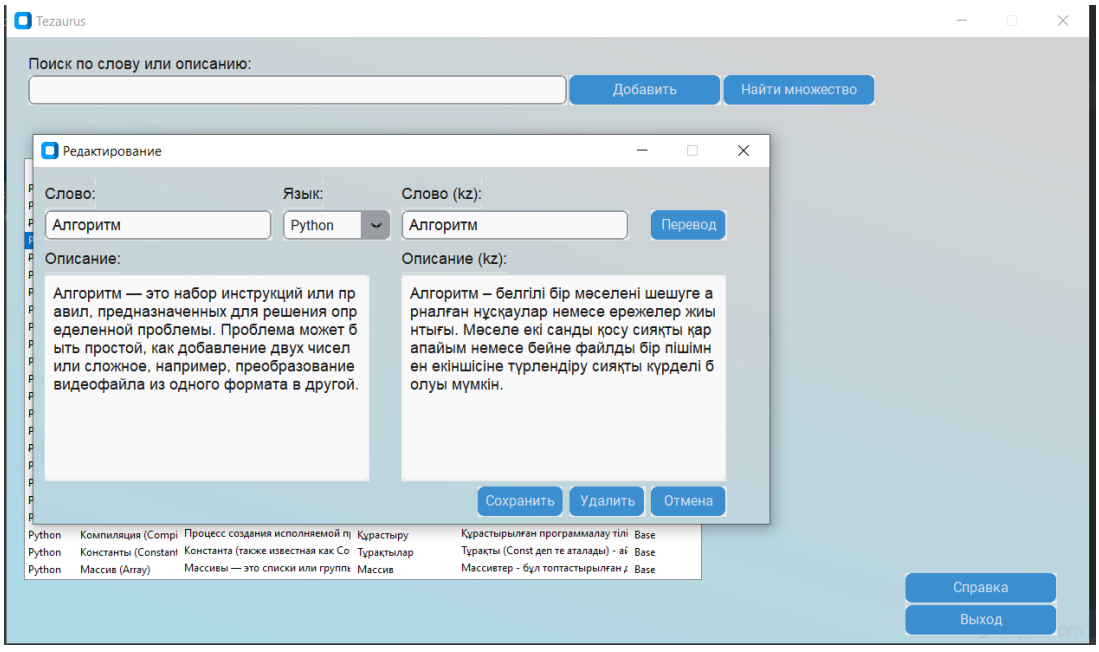


Рисунок 5.2 Управление переводом.

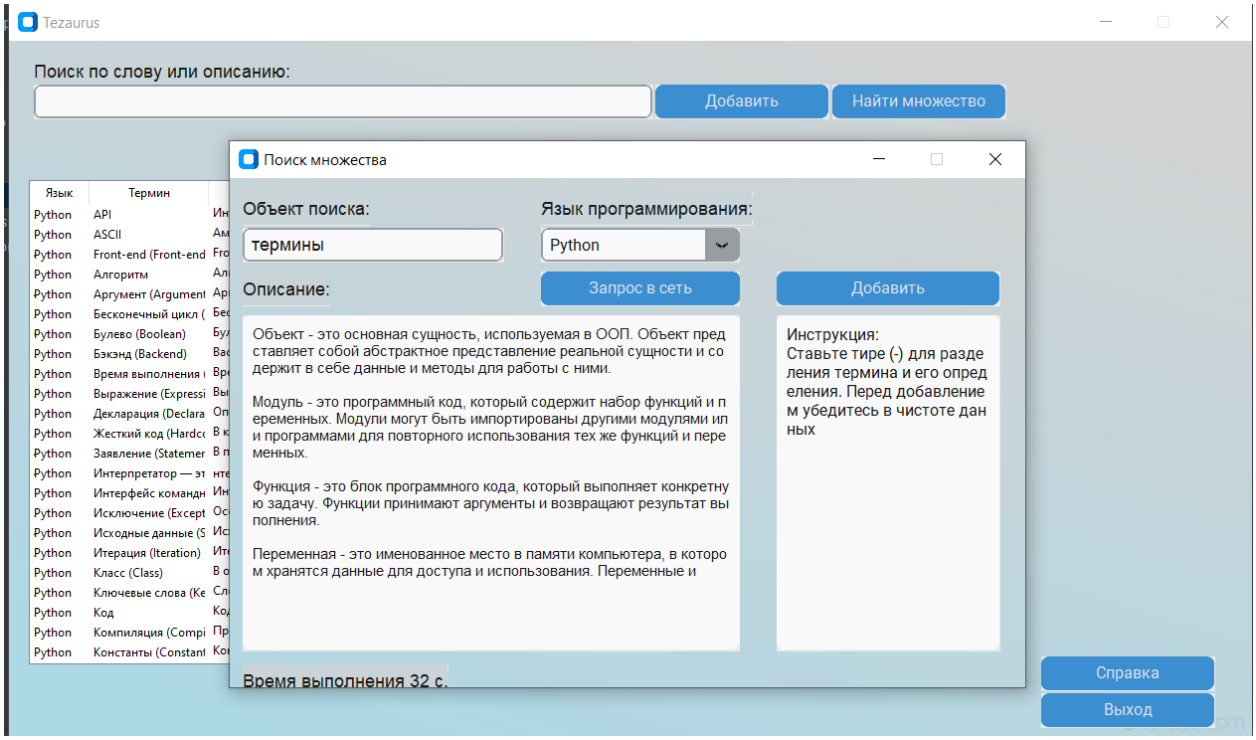


Рисунок 5.3 Общий интерфейс.

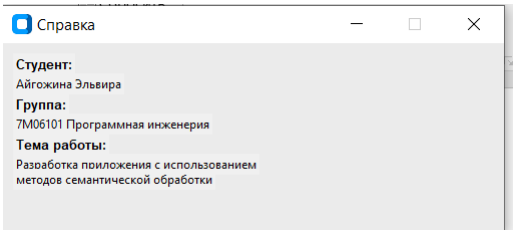


Рисунок 5.4 Справка о студенте.

ЗАКЛЮЧЕНИЕ

Возросло значение методов и технологий семантической обработки в области поиска информации, что позволяет пользователям находить и извлекать соответствующий материал, понимая контекст и значение своих поисковых запросов. Использование тезаурусов, дающих структурированный словарь понятий и отношений для помощи в поиске информации, является основным компонентом семантического поиска. Тезаурусы созданы с учетом набора концепций, обеспечивающих удобство использования и эффективность при поиске информации.

В этой диссертации рассматривалась тема «Разработка приложений с использованием методов семантической обработки». Проблема исследования актуальна, поскольку она соответствует растущему спросу на расширенные методы поиска информации и анализа данных в различных областях. Это исследование вносит свой вклад в тему информационных технологий и их практических приложений, исследуя методы семантической обработки и их использование в разработке программного обеспечения.

Проблема исследования актуальна за пределами академической арены и связана с общественными и национальными программами, направленными на совершенствование науки и техники. Использование методов семантической обработки при разработке приложений помогает стимулировать творчество, повышать доступность информации и совершенствовать процессы принятия решений. Выводы этой диссертации способствуют более широкой национальной миссии по использованию возможностей новых технологий для стимулирования экономического роста, повышения эффективности и обеспечения принятия решений на основе данных во многих секторах.

Цели исследования были четко изложены и эффективно реализованы на протяжении всей диссертации. Тщательное изучение методов семантической обработки, включая их преимущества и недостатки, дает полезную информацию об их потенциале для улучшения поиска информации и анализа данных.

Возникли различные семантические поисковые системы, каждая со своим собственным набором функций и возможностей, например, Google Knowledge Graph, Wolfram Alpha и IBM Watson. Исследователи также опубликовали многочисленные исследования по разработке и применению тезаурусов в информационном поиске. Оценка существующих семантических поисковых систем и тезаурусов улучшила понимание практического использования и влияния на эффективность поиска информации. Строгий подход к обработке и анализу данных привел к созданию тщательно подобранного списка соответствующих терминов и их взаимосвязей, что помогло получить более точную и полезную информацию.

В исследовании также подчеркиваются возможные ограничения современного семантического поиска, такие как проблемы с устранением неоднозначности и необходимость регулярного обновления и поддержки тезаурусов. В то время как диссертация демонстрирует тщательные исследовательские усилия, необходимо признать значительные ограничения в ее

содержании и дизайне. Из-за ограничений по времени и ресурсам эксперимент по созданию тезауруса был ограничен одним доменом и относительно небольшим набором данных. Расширение масштабов будущих исследований для включения большего количества областей и более крупных наборов данных может улучшить обобщаемость и полезность результатов. Кроме того, дополнительное исследование и сравнение альтернативных стратегий семантической обработки может дать более полное представление об их эффективности и применимости в различных обстоятельствах.

С точки зрения научной новизны, эта диссертация дополняет текущий объем знаний, углубляясь в методы семантической обработки и их применение в разработке программного обеспечения. Изучение некоторых тезаурусов, таких как MeSH, APA Thesaurus, ERIC Thesaurus и AGROVOC, позволяет лучше понять их эволюцию, структуру и применение в различных дисциплинах.

Метод исследования, использованный в диссертации, повышает степень обоснованности и надежности каждого открытия, научной точки зрения и вывода, представленных в диссертации. Тщательный анализ экспериментальных результатов, а также надежность используемых статистических подходов обеспечивают правильность и достоверность результатов. Эта диссертация закладывает прочную основу для будущих исследований в области семантической обработки и разработки приложений, придерживаясь передового опыта в методологии исследований.

Результаты этого исследования подчеркивают важность подходов семантической обработки в разработке приложений. Использование тезаурусов и семантических поисковых систем повышает точность, релевантность и эффективность поиска информации, предоставляя пользователям более полные и персонализированные результаты.

В целом, изобретение тезаурусов и их использование в информационном поиске существенно повысили эффективность и точность методов семантического поиска и будут продолжать оставаться важным инструментом для исследователей, профессионалов и всех, кто ищет информацию.

Эта диссертация является отличным ресурсом для исследователей и практиков, заинтересованных в использовании методов семантической обработки в поиске информации и разработке программного обеспечения. Тщательное изучение темы, подкрепленное источниками и строгими исследовательскими методологиями, обеспечивает научную обоснованность и достоверность предлагаемых выводов. Демонстрируя полное понимание темы исследования, предлагая практические идеи и способствуя росту знаний в этой области, диссертация эффективно соответствует требованиям магистерской диссертации. Выводы исследования предлагают четкий и краткий обзор важных выводов, подчеркивая важность подходов семантической обработки для улучшения поиска информации и разработки приложений. Приверженность диссертации стандартам, а также ее научная строгость, практическая значимость и потенциал для будущих разработок делают ее важным дополнением к академической и профессиональной сцене.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Wang Wei, Payam M. Barnaghi, Andrzej Bargiela. Search with Meanings: An Overview of Semantic Search Systems, 2008.
2 Michal Pecánek. “Что такое семантический поиск и как он влияет на SEO?”, 2020.
3 Victoria Uren, Yuangui Lei, Vanessa Lopez, Haiming Liu, Enrico Motta, Marina Giordanino. The usability of semantic search tools: a review,2007.
4 Ricardo Baeza-Yates, Massimiliano Ciaramita, Peter Mika & Hugo Zaragoza. Towards Semantic Search, 2008.
5 Guha R., McCool R., Miller E. Semantic search //Proceedings of the 12th international conference on World Wide Web. – 2003. – С. 700-709.
6 Uren V. et al. The usability of semantic search tools: a review //The Knowledge Engineering Review. – 2007. – Т. 22. – №. 4. – С. 361-377.
7 Vikas Jindal. A review of ranking approaches for semantic search on Web, 2013.
8 Барахнин В. Б., Нехаева В. А. Технология создания тезауруса предметной области на основе предметного указателя энциклопедии, 2007.
9 Bing Liu . Sentiment Analysis: A Survey, 2022.
10 Martha Palmer. A Survey of Semantic Role Labeling, 2021.
11 Palash Goyal.Deep Learning for Natural Language Processing, 2009.
12 Manning, C. D., Raghavan, P., & Schütze, H. Introduction to Information Retrieval. Cambridge University Press, 2008.
13 Jurafsky, D., Martin, J. H. Speech and Language Processing. Pearson Education, 2019.
14 Baeza-Yates, R., Ribeiro-Neto, B. Modern Information Retrieval. ACM Press, 2011.
15 Salton, G., McGill, M. J. Introduction to Modern Information Retrieval. McGraw-Hill, 1983.
16 Sparck Jones, K., Willett, P. Readings in Information Retrieval. Morgan Kaufmann, 1997.
17 Witten, I. H., Moffat, A., Bell, T. C. Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann,2010.
18 Manning, C. D., Raghavan, P., Schütze, H. Introduction to Statistical Natural Language Processing. MIT Press, 2008.
19 Shead T. M. et al. A Novel In Situ Machine Learning Framework for Intelligent Data Capture and Event Detection //Machine Learning and Its Application to Reacting Flows: ML and Combustion. – Cham : Springer International Publishing, 2023. – С. 53-87.
20 Kowalski P. Information retrieval system iSybislaw and a synergic effect: some reflections on terminology and a language of keywords from the linguistic perspective, 2019. – №. 5. – С. 140-147.
21 Croft, W. B., Metzler, D., Strohman, T., Search Engines: Information Retrieval in Practice. Addison-Wesley, 2008.
22 Berry, M. W., Castellanos, M. Survey of Text Mining: Clustering, Classification, and Retrieval. Springer, 2007.

23 Resnik, P., Resnik, R. Semantic Similarity in Natural Language Processing: An Ontology-Based Approach. Morgan & Claypool Publishers, 2012.
24 Manning, C. D., Schütze, H. Foundations of Statistical Natural Language Processing. MIT Press, 1999.
25 Hearst, M. A. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, 2019.
26 Nelson, S. J., Johnston, W. D., & Humphreys, B. L., Relationships in medical subject headings (MeSH). Relationships in the Organization of Knowledge, 2001.-P.171-184
27 Tuleya, L. G. E., Thesaurus of psychological index terms. American Psychological Association, 2007.
28 Hudon M. Multilingual thesaurus construction–integrating the views of different cultures in one gateway to knowledge and concepts //Information services & use. – 1997. – Т. 17. – №. 2-3. – С. 111-123.
29 Owens L. A., Cochrane P. A. Thesaurus evaluation //Cataloging & classification quarterly. – 2004. – Т. 37. – №. 3-4. – С. 87-102.
30 Booth, B., A" New" ERIC Thesaurus, Fine Tuned for Searching. Online, 2001.-P.20-29.
31 Caracciolo, C., Stellato, A., Rajbahndari, S., Morshed, A., Johannsen, G., Jaques, Y., & Keizer, J., Thesaurus maintenance, alignment and publication as linked data: the AGROVOC use case. International Journal of Metadata, Semantics and Ontologies, 2007. 7(1), 65-75.
32 Owens, M., & Allen, G., SQLite. New York: Apress LP, 2007.

```
from tkinter import *
import customtkinter as ct
import tkinter as tk
from tkinter import ttk
from tkinter import Button, Entry, Label
import os
import time
import bot
import dao
import translator
from PIL import Image

win_h = "560"
win_w = "1000"
#Класс главного окна, в котором отображается все данные: словарь, кнопки управления и т.д.
class App(ct.CTk):
   #Метод для инициализации окна
   def __init__(self):
      super().__init__()

      self.title("Tezaurus")
      self.geometry(win_w + "x" + win_h)
      current_path = os.path.dirname(os.path.realpath(__file__))
      self.bg_image = ct.CTkImage(Image.open(current_path + "\\background.jpg"), size=(int(win_w), int(win_h)))
      self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
      self.bg_image_label.place(x=0, y=0)



      def get_by_word(*args):
         app.update_window(db.get_by_word(self.request.get()))

      #Текстовый блок
      search_request_label = ct.CTkLabel(self, text="Поиск по слову или описанию:", font=("Arial", 14), fg_color="#c9d4d8")
      search_request_label.place(x=20, y=10)

      #Поле запроса
      self.request = ct.StringVar()
      self.search_request_field = ct.CTkEntry(self, width=498, textvariable=self.request)
      self.request.trace_add("write", get_by_word)
```

```
        self.search_request_field.place(x=20, y=35)

        #Кнопка Добавить
        add_btn = ct.CTkButton(self, text="Добавить", command=self.open_new)
        add_btn.place(x=520, y=35)
        #Кнопка поиска нескольких терминов
        add_many_btn = ct.CTkButton(self, text="Найти множество",
command=self.open_table)
        add_many_btn.place(x=662, y=35)
        #Кнопка поиска нескольких терминов
        exit_btn = ct.CTkButton(self, text="Выход", command=self.destroy)
        exit_btn.place(x=830, y=525)
        #Кнопка Справки
        exit_btn = ct.CTkButton(self, text="Справка", command=self.open_help)
        exit_btn.place(x=830, y=495)


        #Таблица результатов
        table_columns = ("id", "lang","word","definition", "kk_title",
"kk_definition","owner")
        self.table = ttk.Treeview(columns=table_columns, show="headings",
height=23)
        self.table.bind("<Double-1>", self.onDoubleClick)

        #Определяем заголовки
        self.table.heading("id", text="№")
        self.table.heading("lang", text="Язык")
        self.table.heading("word", text="Термин")
        self.table.heading("definition", text="Описание")

        self.table.heading("kk_title", text="Термин (kz)")
        self.table.heading("kk_definition", text="Описание (kz)")
        self.table.heading("owner", text="Добавлено")

        self.table.column("#1", stretch='no', width=0)
        self.table.column("#2", stretch='no', width=60)
        self.table.column("#3", stretch='no', width=120)
        self.table.column("#4", stretch='no', width=200)
        self.table.column("#5", stretch='no', width=120)
        self.table.column("#6", stretch='no', width=200)
        self.table.column("#7", stretch='no', width=80)
        # tct.Style().configure('Treeview', rowheight=30)
        self.table.place(x=20, y=140, anchor="nw")

        #Начальное заполнение таблицы
```

```
        self.update_window(db.get_all())


    def onDoubleClick(self, event):
        select = app.table.selection()
        item = self.table.item(select)
        self.open_edit()


    def open_edit(self):
        if (self.table.focus() != ""):
            edit = Edit(self)
            edit.grab_set()

    def open_new(self):
        new = New(self)
        new.grab_set()

    def open_table(self):
        table1 = Term_table(self)
        table1.grab_set()

    def update_window(self, data):
        self.table.delete(*self.table.get_children())
        for line in data:
            self.table.insert("", "end", values=line)

    def delete_item(self):
        if (self.table.focus() != ""):
            id = app.table.item(app.table.focus()).get("values")[0]
            db.remove(id)
            self.update_window(db.get_all())

    def open_help(self):
        info = Info(self)
        info.grab_set()

# #Класс для работы с окном редактирования
class Edit(ct.CTkToplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.title("Редактирование")
        self.resizable(width=False, height=False)
        self.geometry('680x330')
        current_path = os.path.dirname(os.path.realpath(__file__))
```

```
        self.bg_image = ct.CTkImage(Image.open(current_path + "\\background.jpg"),size=(int(win_w), int(win_h)))
        self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
        self.bg_image_label.place(x=0, y=0)

        self.word_to_change = ct.StringVar()

self.word_to_change.set(app.table.item(app.table.focus()).get("values")[2])

        self.kk_to_change = ct.StringVar()
        self.kk_to_change.set(app.table.item(app.table.focus()).get("values")[4])
        #Блок для редактирования слова
        word_edit_label = ct.CTkLabel(self, text="Слово:", font=("Arial", 14), fg_color="#c9d4d8")
        word_edit_label.place(x=10, y=10)
        kk_edit_label =ct.CTkLabel(self, text="Слово (kz):", font=("Arial", 14), fg_color="#c9d4d8")
        kk_edit_label.place(x=340, y=10)
        #Позиционирование элементов ввода слова
        self.word_edit_field =ct.CTkEntry(self, width=210, textvariable=self.word_to_change, font=("Arial", 14))
        self.word_edit_field.place(x=10, y=40)
        self.word_edit_field.focus()

        self.kk_edit_field =ct.CTkEntry(self, width=210, textvariable=self.kk_to_change, font=("Arial", 14))
        self.kk_edit_field.place(x=340, y=40)


        # Создание надписи и поля ввода
        lang_edit_label =ct.CTkLabel(self, text="Язык:", font=("Arial", 14), fg_color="#c9d4d8")
        lang_edit_label.place(x=230, y=10)

        #Позиционирование элементов ввода слова
        self.lang_to_change = ct.StringVar()
        self.lang_to_change.set(app.table.item(app.table.focus()).get("values")[1])
        self.lang_edit_field = ct.CTkComboBox(self, values=db.get_langs(), width=100, variable=self.lang_to_change)
        self.lang_edit_field.place(x=230, y=40)


        #Панель описания
        word_definition_label =ct.CTkLabel(self, text="Описание:", font=("Arial", 14), fg_color="#c9d4d8")
```

```
            word_definition_label.place(x=10, y=70)
            kk_definition_label =ct.CTkLabel(self, text="Описание (kz):",
font=("Arial", 14), fg_color="#c9d4d8")
            kk_definition_label.place(x=340, y=70)

            self.word_definition = ct.CTkTextbox(self, width=300, height=190)
            self.word_definition.configure(font=("Arial", 14))
            self.word_definition.insert(1.0,
app.table.item(app.table.focus()).get("values")[3])
            self.word_definition.place(x=10, y=100)

            self.kk_word_definition = ct.CTkTextbox(self, width=300, height=190)
            self.kk_word_definition.configure(font=("Arial", 14))
            self.kk_word_definition.insert(1.0,
app.table.item(app.table.focus()).get("values")[5])
            self.kk_word_definition.place(x=340, y=100)

            #Управление словарем
            save_btn = ct.CTkButton(self, text="Сохранить",
command=self.update_record, width=70)
            save_btn.place(x=410, y=295)

            delete_btn = ct.CTkButton(self, text="Удалить", command=self.delete,
width=70)
            delete_btn.place(x=495, y=295)

            cancel_btn = ct.CTkButton(self, text="Отмена", command=self.destroy,
width=70)
            cancel_btn.place(x=570, y=295)

            translate_btn = ct.CTkButton(self, text="Перевод",
command=self.translate_kk, width=70)
            translate_btn.place(x=570, y=40)

        def delete(self):
            app.delete_item()
            self.destroy()

        def translate_kk(self):
            self.kk_to_change.set(translator.translate(self.word_to_change.get()))
            self.kk_word_definition.delete(1.0, ct.END)
            self.kk_word_definition.insert(1.0,
translator.translate(self.word_definition.get(1.0, ct.END)))

        def update_record(self):
```

```
            id = app.table.item(app.table.focus()).get("values")[0]
            word = self.word_edit_field.get()
            definition = self.word_definition.get(1.0, 'end')
            db.update(id, self.lang_to_change.get(), word, definition,
self.kk_edit_field.get(), self.kk_word_definition.get(1.0, ct.END))
            app.update_window(db.get_all())
            self.destroy()

        # #Класс окна для добавления нового элемента в словарь
        class New(ct.CTkToplevel):

            # Инициализация окна
            def __init__(self, parent):
                super().__init__(parent)
                self.owner = 'User'
                self.title("Новое слово")
                self.geometry('400x330')
                self.resizable(width=False, height=False)
                current_path = os.path.dirname(os.path.realpath(__file__))
                self.bg_image = ct.CTkImage(Image.open(current_path +
"\\background.jpg"),size=(int(win_w), int(win_h)))
                self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
                self.bg_image_label.place(x=0, y=0)
                word_to_add = ct.StringVar()
                word_to_add.set(app.request.get())

                # Создание надписи и поля ввода
                word_edit_label = ct.CTkLabel(self, text="Слово:", font=("Arial", 14),
fg_color="#c9d4d8")
                word_edit_label.place(x=10, y=5)

                #Позиционирование элементов ввода слова
                self.word_edit_field = ct.CTkEntry(self, width=200, font=("Arial", 14),
textvariable=word_to_add)
                self.word_edit_field.place(x=10, y=30)
                self.word_edit_field.focus()

                # Создание надписи и поля ввода
                lang_edit_label = ct.CTkLabel(self, text="Язык:", font=("Arial", 14),
fg_color="#c9d4d8")
                lang_edit_label.place(x=230, y=5)

                #Позиционирование элементов ввода слова
                self.lang_edit_field = ct.CTkComboBox(self, values=db.get_langs())
                self.lang_edit_field.place(x=230, y=30)
```

```
            #Панель описания
            word_definition_label = ct.CTkLabel(self, text="Описание:",
font=("Arial", 14), fg_color="#c9d4d8")
            word_definition_label.place(x=10, y=60)


            self.word_definition = ct.CTkTextbox(self, width=370, height=200)
            # self.word_definition.configure(font=("Arial", 10))
            self.word_definition.place(x=10, y=85)

            #Управление словарем
            search_btn = ct.CTkButton(self, text="Запрос в сеть",
command=self.ask_network, width=80)
            search_btn.place(x=100, y=295)

            save_btn = ct.CTkButton(self, text="Сохранить",
command=self.add_to_db, width=80)
            save_btn.place(x=210, y=295)

            cancel_btn = ct.CTkButton(self, text="Отмена", command=self.destroy,
width=80)
            cancel_btn.place(x=300, y=295)

        def add_to_db(self):
            db.add(self.lang_edit_field.get()                ,self.word_edit_field.get(),
self.word_definition.get("1.0", "end"), self.owner)
            app.update_window(db.get_all())
            self.destroy()

        def ask_network(self):
            self.owner = "Bot"
            response = bot.ask(self.word_edit_field.get())
            self.word_definition.delete(1.0, ct.END)
            self.word_definition.insert(1.0, response)

    #Модуль импорта нескольких терминов
    class Term_table(ct.CTkToplevel):
        def __init__(self, parent):
            super().__init__(parent)
            self.owner = 'User'
            self.title("Поиск множества")
            self.resizable(width=False, height=False)
            self.geometry('640x410')
```

```
        current_path = os.path.dirname(os.path.realpath(__file__))
        self.bg_image = ct.CTkImage(Image.open(current_path + "\\background.jpg"),size=(int(win_w), int(win_h)))
        self.bg_image_label = ct.CTkLabel(self, image=self.bg_image)
        self.bg_image_label.place(x=0, y=0)

        #Блок для редактирования слова
        term_edit_label =ct.CTkLabel(self, text="Объект поиска:", font=("Arial", 14), fg_color="#c9d4d8")
        term_edit_label.place(x=10, y=10)

        #Позиционирование элементов ввода слова
        self.term_edit_field =ct.CTkEntry(self, width=210, font=("Arial", 14))
        self.term_edit_field.place(x=10, y=40)
        self.term_edit_field.focus()

        #Панель описания
        lang_label =ct.CTkLabel(self, text="Язык программирования:", font=("Arial", 14), fg_color="#c9d4d8")
        lang_label.place(x=250, y=10)

        self.time_var = ct.StringVar()
        self.time_var.set("")
        self.time_label =ct.CTkLabel(self, textvariable=self.time_var, font=("Arial", 14), fg_color="#c9d4d8")
        self.time_label.place(x=10, y=390)

        self.lang_edit_field = ct.CTkComboBox(self, values=db.get_langs(), width=160)
        self.lang_edit_field.place(x=250, y=40)



        definition_label =ct.CTkLabel(self, text="Описание:", font=("Arial", 14), fg_color="#c9d4d8")
        definition_label.place(x=10, y=75)


        self.definition = ct.CTkTextbox(self, width=400, height=270)
        self.definition.configure(font=("Arial", 12))
        self.definition.place(x=10, y=110)

        info_text = '''Инструкция:\nСтавьте тире (-) для разделения термина и его определения. Перед добавлением убедитесь в чистоте данных'''
```

```
            search_btn = ct.CTkButton(self, text="Запрос в сеть",
command=self.ask_network,  width=160)
            search_btn.place(x=250, y=75)
            add_btn = ct.CTkButton(self, text="Добавить", command=self.add_to_db,
width=180)
            add_btn.place(x=440, y=75)
            info_label = ct.CTkTextbox(self, width=180, height=270)
            info_label.place(x=440, y=110)
            info_label.insert(1.0, info_text)

        def add_to_db(self):
            raw_data_list = self.definition.get("1.0",ct.END).split("\n")
            term_list = []

            for item in raw_data_list:
                if item != "":
                    term_list.append([item[:item.find(' - ')], item[item.find(' - ')+2:]])
            for item in term_list:
                db.add(self.lang_edit_field.get(), item[0], item[1], self.owner)
            app.update_window(db.get_all())
            self.destroy()

        def ask_network(self):
            time1 = time.time()
            response = bot.ask(self.term_edit_field.get(), self.lang_edit_field.get())
            time2 = time.time()
            self.owner = "Bot"
            self.time_var.set("Время выполнения {0} с.".format(round(time2-
time1)))
            self.definition.delete(1.0, ct.END)
            self.definition.insert(1.0, response)

    #Окно информации о студенте
    class Info(ct.CTkToplevel):
        def __init__(self, parent):
            super().__init__(parent)
            self.title("Справка")
            self.geometry('400x150')
            self.resizable(width=False, height=False)

            Label(self, text="Студент:", font="Arial 10 bold").place(x = 10, y = 10)
            Label(self, text="Айгожина Эльвира").place(x = 10, y = 30)

            Label(self, text="Группа:", font="Arial 10 bold").place(x = 10, y = 50)
```

```
        Label(self, text="7M06101 Программная инженерия").place(x = 10, y = 70)

        Label(self, text="Тема работы:", font="Arial 10 bold").place(x = 10, y = 90)
        Label(self, text="Разработка приложения с использованием", anchor='w' ).place(x = 10, y = 110)
        Label(self, text="методов семантической обработки", anchor='w' ).place(x = 10, y = 125)

    if __name__ == "__main__":
        db = dao.DAO()
        app = App()
        ct.set_appearance_mode("light")
        ct.set_default_color_theme("blue")
        app.resizable(width=False, height=False)
        app.mainloop()
```

```
import sqlite3 as sl
import os


class DAO:
    def __init__(self):
        self.connection = ''
        self.add_to_base_sql = 'INSERT INTO DICTIONARY (lang, word, definition, owner) values (?, ?, ?, ?)'
        self.update_sql = 'UPDATE DICTIONARY SET word = ?, definition = ?, lang = ?, kk_title = ?, kk_definition = ? where id = ?;'
        self.tmp_word=''
        self.select_from_db_by_word_sql = "SELECT * FROM DICTIONARY WHERE word LIKE ? OR definition LIKE ? OR lang LIKE ? ORDER by word"
        self.select_all_from_db = 'SELECT * FROM DICTIONARY ORDER by word'
        self.delete_from_db_by_id_sql = 'DELETE FROM DICTIONARY WHERE id = ?'
        self.file_name = "dictionary.db"

        if os.path.isfile("dictionary.db"):
            self.connection = sl.connect('dictionary.db')
        else:
            self.connection = sl.connect('dictionary.db')
            with self.connection:
                self.connection.execute(
                    """
                        CREATE TABLE DICTIONARY (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            lang TEXT DEFAULT "Python",
                            word TEXT,
                            definition TEXT,
                            kk_title TEXT DEFAULT "",
                            kk_definition TEXT DEFAULT "",
                            owner TEXT DEFAULT "Base"
                        );
                    """
                )
                for line in db_init_data:
                    self.connection.execute("INSERT INTO DICTIONARY (word, definition) values (?, ?)", line)

    def get_langs(self):
        cur =  self.connection.cursor()
```

```
            cur.execute("SELECT DISTINCT lang FROM DICTIONARY")
            responce = cur.fetchall()
            answer = []
            for item in responce:
                answer.append(str(item)[2:-3])
            return answer

        def add(self, lang, word, definition, owner):
            data = (lang, word, definition, owner)
            cur =  self.connection.cursor()
            cur.execute(self.add_to_base_sql, data)
            self.connection.commit()

        def update(self, id, lang, word, definition, kk_title, kk_definition):
            data = (word, definition, lang, kk_title, kk_definition,id)
            cur =  self.connection.cursor()
            cur.execute(self.update_sql, data)
            self.connection.commit()

        def remove(self, id):
            data = (id,)
            cur =  self.connection.cursor()
            cur.execute(self.delete_from_db_by_id_sql, data)
            self.connection.commit()

        def get_all(self):
            cur =  self.connection.cursor()
            cur.execute(self.select_all_from_db)
            responce = cur.fetchall()
            return responce

        def get_by_word(self, word):
            cur =  self.connection.cursor()
            cur.execute(self.select_from_db_by_word_sql, ("%" + word + "%", "%" +
word + "%", "%" + word + "%",))
            responce = cur.fetchall()
            return responce
```

```
        from deep_translator import GoogleTranslator

def translate(text):
    translated = GoogleTranslator(source="ru", target="kk").translate(text=text)
    return translated

        import openai


openai.api_key = 'sk-
yQnqg9ZHL4PQGp7Ovz7BT3BlbkFJLTHOk3czwn1oAAjuXHld'
engine = 'text-davinci-003'

def ask(prompt, lang=''):
    prompt = prompt + ' ' + lang
    completion = openai.Completion.create(engine=engine,
                                          prompt=prompt,
                                          temperature=0.5,
                                          max_tokens=700)
    return completion.choices[0]['text'][:]
```