

Реферат

Расчетно-пояснительная записка 33 с., 5 рис., 0 табл., ? ист., ? прил.

Ключевые слова: научно-технические тексты, многокомпонентный термин, извлечение терминов, базы данных, PostgreSQL, NoSQL, Redis, gRPC, кэширование данных.

Цель работы: разработка системы извлечения многокомпонентных терминов и их переводных эквивалентов из параллельных научно-технических текстов.

Разработанное ПО является приложением для выделения терминов из текстов, их сохранения и анализа. Приложение может использоваться лингвистами для сбора терминологических баз данных и проведения исследований.

СОДЕРЖАНИЕ

Введение	6
1 Аналитическая часть	8
1.1 Метод извлечения многокомпонентных терминов на основе структурных моделей	8
1.2 Постановка задачи	9
1.3 Формализация ролей	9
1.4 Формализация данных	12
1.5 Анализ моделей БД	14
1.5.1 Классификация СУБД по модели данных	14
1.5.2 Выбор модели БД	16
2 Конструкторская часть	18
2.1 Проектирование БД	18
2.1.1 Формализация сущностей системы	18
2.1.2 Триггеры БД?	18
2.1.3 Ролевая модель	18
2.2 Проектирование программного комплекса	18
2.2.1 Архитектура приложения	18
2.2.1.1 Монолитная архитектура	18
2.2.1.2 Микросервисная архитектура	19
2.2.1.3 Выбор архитектуры приложения	19
2.2.2 Связь компонентов приложения	19
2.2.2.1 REST API	19
2.2.2.2 gRPC	19
2.2.2.3 Выбор взаимодействия компонентов приложения	19
2.2.3 Структура программного комплекса	19
2.2.4 Диаграммы последовательностей	20
2.2.5 Паттерны проектирования	20
2.2.5.1 Repository	20
2.2.5.2 Dependency injection	21
2.2.5.3 Template method	21

3	Технологическая часть	23
3.1	Анализ СУБД	23
3.1.1	SQLite?	23
3.1.2	MySQL	23
3.1.3	Oracle	23
3.1.4	Microsoft SQL Server	23
3.1.5	PostgreSQL	23
3.1.6	MongoDB	23
3.1.7	Redis	23
3.1.8	Выбор СУБД для решения задачи	23
3.2	Средства реализации	23
3.3	Детали реализации	23
3.3.1	Триггеры БД?	23
3.3.2	Роли БД	23
3.3.3	Примеры работы ПО	23
4	Экспериментально-исследовательская часть	24
4.1	Цель проводимых измерений	24
4.2	Описание проводимых измерений	24
4.3	Инструменты измерения времени работы программы	24
4.4	Результаты проведённых измерений	24
	Заключение	25
	Список использованных источников	26
	Приложение А	29
	Приложение Б	33

Определения, обозначения и сокращения

Модель данных — это абстрактное, независимое, логическое определение структур данных, операторов над данными и прочего, что в совокупности составляет абстрактную систему, с которой взаимодействует пользователь.

База данных (БД) — совокупность взаимосвязанных данных некоторой предметной области, хранимых в памяти ЭВМ и организованных таким образом, что эти данные могут быть использованы для решения многих задач многими пользователями.

Система управления базами данных (СУБД) — приложение, обеспечивающее создание, хранение, обновление и поиск информации в базах данных.

Индекс — объект базы данных, создаваемый с целью повышения производительности поиска данных.

Сервис — абстракция, определяющая что-то, что предоставляет услугу.

Компонент — реализация сервиса, которая содержит поведение.

Микросервис — сервис, отвечающий за один элемент логики в определенной предметной области.

Введение

Стремительное развитие и внедрение технологий искусственного интеллекта и технологий автоматической обработки текстовой информации способствуют развитию лингвистических баз данных как основы создания прикладных программных средств, проведения лингвистических исследований источников информации при решении ряда прикладных задач, где однозначная и упорядоченная терминология имеет особую значимость. Под терминологической базой данных принято понимать организованную в соответствии с определёнными правилами и поддерживаемую в памяти компьютера совокупность данных, характеризующую актуальное состояние некоторой предметной области и используемую для удовлетворения информационных потребностей пользователей [1]. Каждый термин дополнен информацией о его значении, эквивалентах в других языках, кратких формах, синонимах, сведениях об области применения. По целевому назначению терминологические базы данных разделяют на одноязычные, предназначенные для обеспечения информацией о стандартизованной и рекомендованной терминологии, и многоязычные, ориентированные на работы по переводу научно-технической литературы и документации [2] [3].

Создание терминологических баз данных представляет собой сложный и трудоемкий процесс, требующий значительного количества времени на их создание и обновление, что особенно важно для развивающихся терминологий таких предметных областей, как авиация, космонавтика, нанотехнологии, биоинженерия, информационные технологии и многих других. Одним из наиболее время-затратных процессов является ручной сбор иллюстративного материала – извлечение специальной терминологии из коллекций текстов, что требует наличия средств автоматического извлечения многокомпонентных терминов при обработке научно-технических текстов.

Существующие программные средства автоматического извлечения терминов основаны на лингвистических и статистических методах. Могут быть ис-

пользованы и методы машинного обучения [4], сложность их реализации вызвана необходимостью наличия огромных массивов обучающих данных, которые могут отсутствовать для определенной предметной области. В основе лингвистических методов лежит использование грамматики лексико-синтаксических шаблонов, представляющих собой структурные модели лингвистических конструкций [5] [6]. Статистический подход заключается в нахождении n-грамм слов по заданным частотным характеристикам [7] [8]. Гибридный подход для выделения терминологических сочетаний, объединяющий лингвистический и статистический методы, заключается в предварительном описании моделей, по которым могут быть построены термины для последующего поиска их в коллекции текстов [9].

Выравнивание терминологических единиц в параллельных текстах обычно осуществляется в два этапа: сначала выделяют терминологические единицы на каждом языке отдельно, затем один из одноязычных списков терминов-кандидатов интерпретируется как язык источника, и для каждого термина-кандидата на языке источнике предлагаются потенциально эквивалентные термины в списке терминов-кандидатов на языке перевода [10].

Целью данной работы является разработка системы извлечения многокомпонентных терминов и их переводных эквивалентов из параллельных научно-технических текстов. Для достижения поставленной цели необходимо решить следующие задачи.

1. Провести анализ предметной области и формализовать задачу.
2. Спроектировать базу данных и структуру программного обеспечения.
3. Реализовать интерфейс для доступа к базе данных.
4. Реализовать программное обеспечение, которое позволит пользователю создавать, получать и изменять сведения из разработанной базы данных.
5. Провести исследование зависимости времени выполнения запросов от использования кеширования данных текущей сессии пользователя.

1 Аналитическая часть

В данном разделе будет приведена постановка задачи, описана модель и структура базы данных, а также будут определены роли пользователей в системе.

1.1 Метод извлечения многокомпонентных терминов на основе структурных моделей

Предлагаемый метод автоматического извлечения русскоязычных многокомпонентных терминов на основе базы данных структурных моделей терминологических словосочетаний состоит из пяти основных этапов [11].

1. Анализ предложения по частям речи.
2. Удаление частей речи и их сочетаний, которые не входят в состав терминологических словосочетаний:
 - глаголы;
 - союзы;
 - местоимения;
 - частицы;
 - знаки препинания;
 - «наречие + предлог».
3. Удаление из оставшихся терминов-кандидатов стоп-слов, указанных в специальной зоне словаря. Под стоп-словами понимаются слова, которые образуют широко используемые коллокации с терминами, но в совокупности не являются терминами по сути, например «современная химия», «рассматриваемый метод синтеза о-гликозидов».
4. Соотнесение полученных цепочек слов с шаблонами терминологических словосочетаний, которые хранятся в базе структурных моделей терминов.
5. Проверка полученных терминов-кандидатов по словарю корпуса.
 - 5.1. Если термин-кандидат есть в словаре, то он извлекается как термин.
 - 5.2. Если полученный термин-кандидат отсутствует в словаре, то он отправ-

ляется терминологу для ручной обработки.

- 5.3. Если термин-кандидат состоит из нескольких слов, то производится попытка разбить его на несколько терминов.

1.2 Постановка задачи

Диаграмма, оформленная в соответствии с нотацией IDEF0 и отражающая декомпозицию алгоритма работы системы извлечения многокомпонентных терминов, представлена в приложении А. Исходя из специфики решаемой задачи, можно сформулировать требования к приложению [12].

1. Задание исходных текстов должно производиться из файла или через специально предусмотренные поля ввода.
2. Пользователю должна быть предоставлена возможность перед сохранением терминов в базу данных выполнять над ними следующие операции.
 - 2.1. Редактирование.
 - 2.2. Сопоставление переводных эквивалентов.
 - 2.3. Присвоение характеристик.
3. Пользователь может анализировать и редактировать добавленные в базу данных термины путём их поиска по заданным характеристикам.

1.3 Формализация ролей

Для работы с системой обязательным этапом является прохождение аутентификации. Пользователь может работать в системе под одной из следующих ролей.

1. Студент — пользователь, имеющий возможность обрабатывать тексты, сохранять выделенные термины, а также анализировать и редактировать только те термины, которые он выделил.
2. Преподаватель — пользователь, обладающий функционалом, доступным роли «Студент», а также имеющий возможность анализировать и редактировать все термины, хранящиеся в базе данных.
3. Администратор — пользователь, имеющий возможности, доступные роли «Преподаватель», а также имеющий в распоряжении специальные функ-

ции для анализа состояния системы и настройки её работы. Также администратор имеет возможность создавать аккаунты, соответствующие любой из ролей в системе.

В ходе использования приложения должна быть предусмотрена возможность смены ролей путём прохождения повторной аутентификации.

Также стоит отметить, что до входа в аккаунт пользователь считается неавторизованным и не имеет доступа к функционалу системы, так как любая работа с терминами должна быть персонализирована.

На рисунке 1 представлена диаграмма вариантов использования системы в соответствии с выделенными типами пользователей.

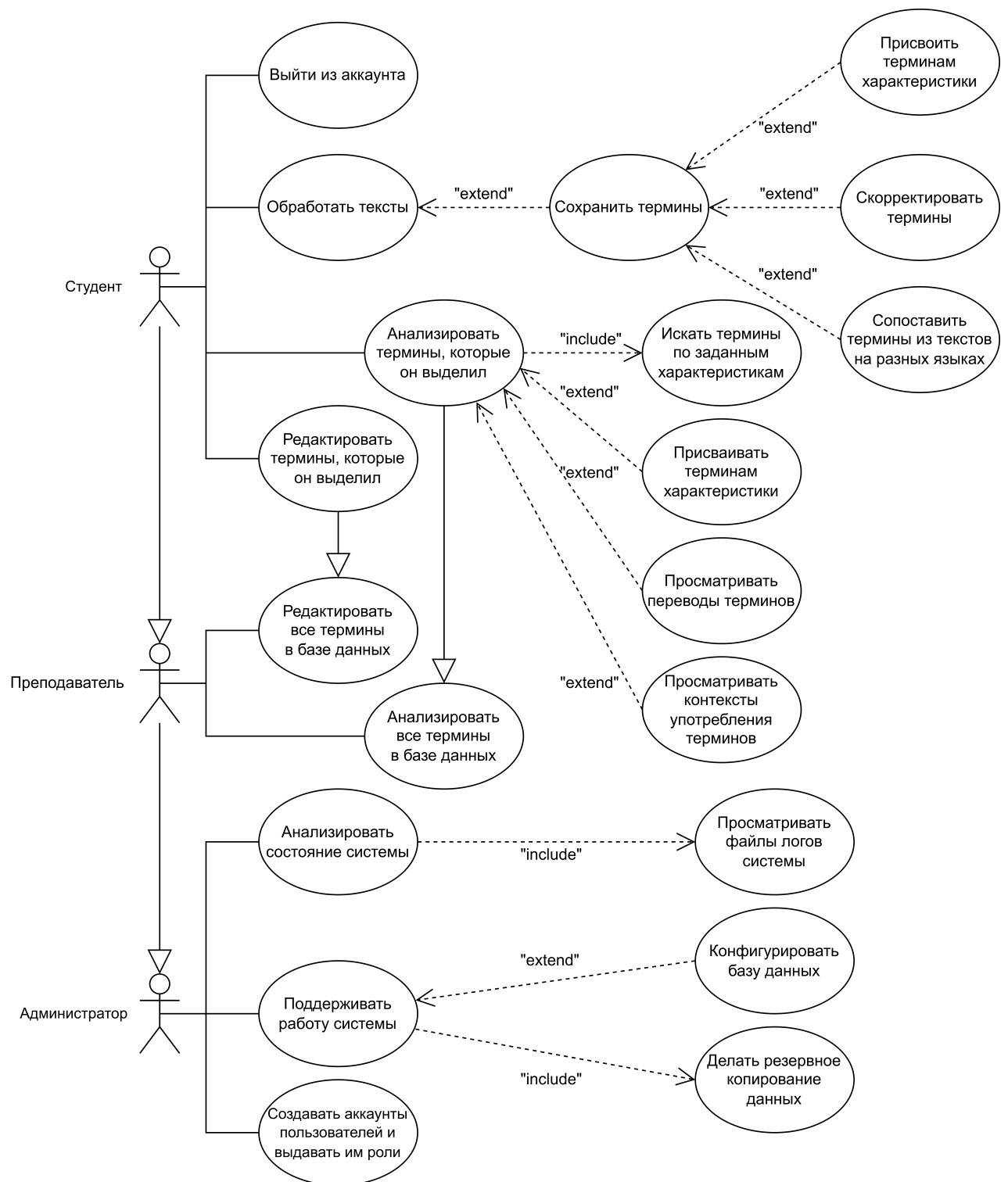


Рисунок 1 – Диаграмма вариантов использования

1.4 Формализация данных

Разрабатываемая база данных должна хранить информацию о следующих сущностях:

- пользователи;
- русскоязычные термины;
- англоязычные термины;
- характеристики терминов;
- структурные модели терминов;
- части речи;
- контексты употребления терминов.

На рисунке 2 представлена ER-диаграмма сущностей в нотации Чена, показывающая сущности, их атрибуты и связи между сущностями в разрабатываемой базе данных.

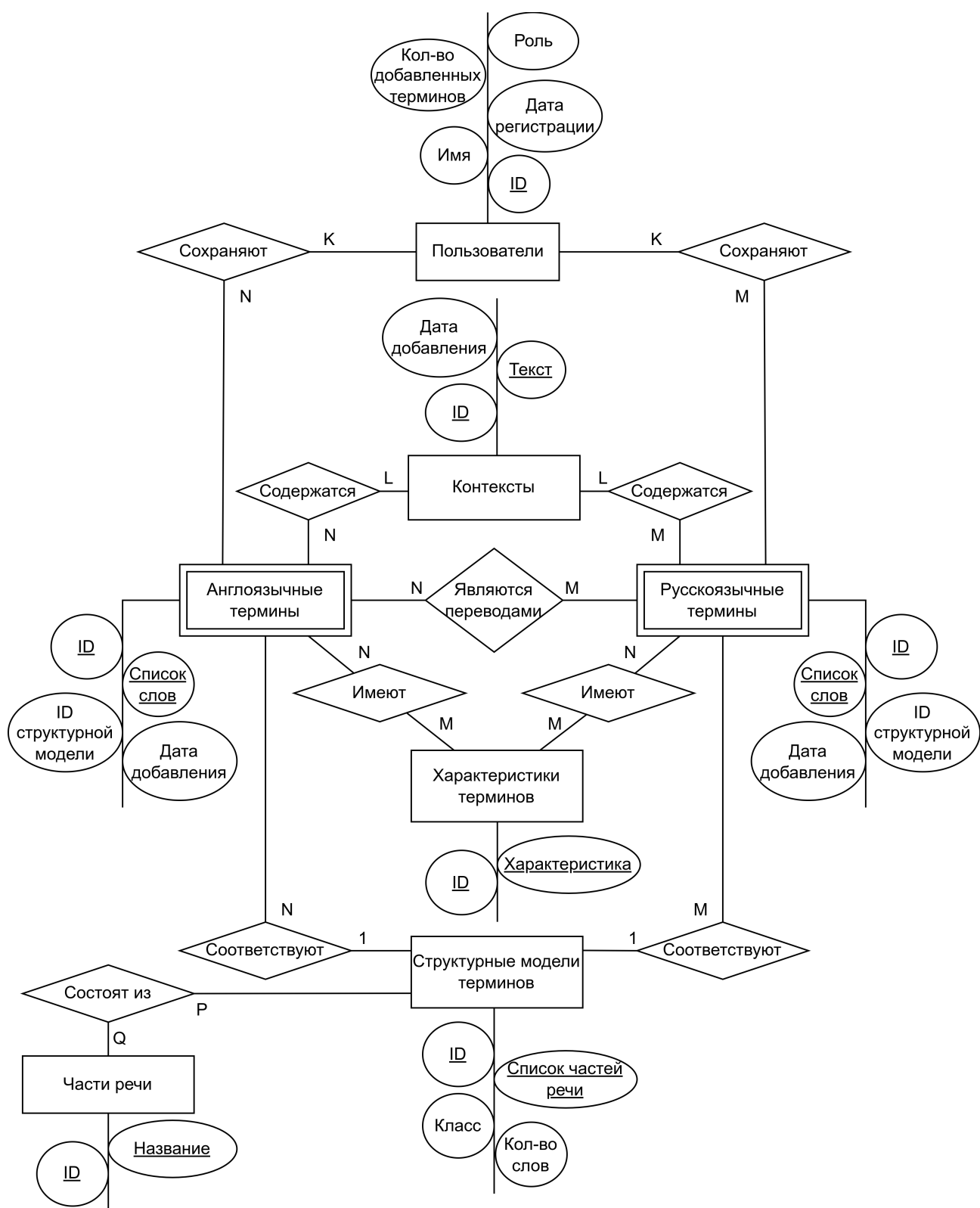


Рисунок 2 – ER-диаграмма сущностей базы данных

1.5 Анализ моделей БД

1.5.1 Классификация СУБД по модели данных

По модели данных СУБД можно разделить на следующие типы.

1. Дореляционные.

Старые (дореляционные) системы можно разделить на три большие категории: системы с инвертированными списками (inverted list), иерархические (hierarchical) и сетевые (network) [14].

1.1. Инвертированные списки (файлы).

БД на основе инвертированных списков представляет собой совокупность файлов (таблиц), содержащих записи. Для записей в файле определен некоторый порядок, диктуемый физической организацией данных. Для каждого файла может быть определено произвольное число других упорядочений на основании значений некоторых полей записей (инвертированных списков). Обычно для этого используются индексы. В такой модели данных отсутствуют ограничения целостности как таковые. Все ограничения на возможные экземпляры БД задаются теми программами, которые работают с БД. Одно из немногих ограничений, которое может присутствовать — это ограничение, задаваемое уникальным индексом.

1.2. Иерархические.

Иерархическая модель БД состоит из объектов с указателями от родительских объектов к дочерним, соединяя вместе связанную информацию. Иерархические БД могут быть представлены в виде дерева. Их производительность в значительной степени зависит от подхода, выбранного самим пользователем (прикладным программистом и/или администратором базы данных).

1.3. Сетевые.

К основным понятиям сетевой модели БД относятся элемент (узел) и

связь. Узел — это совокупность атрибутов данных, описывающих некоторый объект. Сетевые БД могут быть представлены в виде графа. В сетевой БД логика процедуры выборки данных зависит от физической организации этих данных. Поэтому эта модель не является полностью независимой от приложения. Другими словами, если необходимо изменить структуру данных, то нужно изменить и приложение.

2. Реляционные.

Ключевые особенности реляционной модели данных [14]:

- Определение реляционной системы требует, чтобы база данных только воспринималась пользователем как набор таблиц. Таблицы в реляционной системе являются логическими, а не физическими структурами. Таблицы представляют собой абстракцию способа физического хранения данных, в которой детали реализации на уровне физической памяти скрыты от пользователя.
- Информационный принцип: все информационное наполнение базы данных представлено одним и только одним способом, а именно — явным заданием значений, помещенных в позиции столбцов в строках таблицы. Этот метод представления — единственно возможный для реляционных баз данных. В частности, нет никаких указателей, связывающих одну таблицу с другой.

Реляционная модель состоит из пяти компонентов [14].

- Неограниченный набор скалярных типов (включая, в частности, логический тип).
- Генератор типов отношений и соответствующая интерпретация для сгенерированных типов отношений.
- Возможность определения переменных отношения для указанных сгенерированных типов отношений.
- Операция реляционного присваивания для присваивания реляционных значений указанным переменным отношения.

- Неограниченный набор общих реляционных операторов (реляционная алгебра) для получения значений отношений из других значений отношений.

3. Постреляционные.

В связи с быстрым ростом количества данных и их усложнением возникла необходимость в поиске новых подходов к хранению и обработке, отличных от реляционных. Таким решением стала NoSQL-технология (Not Only SQL). Постреляционная модель является расширением реляционной модели. Она снимает ограничение неделимости данных, допуская многозначные поля, значения которых не являются атомарными, и набор значений воспринимается как самостоятельная таблица, встроенная в главную таблицу [15].

Наиболее популярными типами РаБД являются:

- ключ-значение (Redis, Tarantool, Oracle NoSQL DB);
- колоночные (Vertica, ClickHouse, HBase);
- документо-ориентированные (CouchDB, MongoDB);
- графовые (InfoGrid, GraphX, Neo4j).

1.5.2 Выбор модели БД

Для долговременного хранения данных будет использоваться реляционная модель по следующим причинам:

- 1) данные имеют чётко заданную структуру;
- 2) исключается дублирование данных за счёт использования связей между отношениями с помощью внешних ключей;
- 3) доступ к данным отделяется от способа их организации на уровне физической памяти;

Для кратковременного хранения данных о текущей сессии пользователя (в частности, сохранённых им терминов) будет использоваться нереляционная модель по следующим причинам:

- 1) данные могут не иметь общей структуры;

- 2) появляется возможность хранения вложенных структур данных;
- 3) повышается быстродействие за счёт возможности хранения всех данных в оперативной памяти (In-Memory DataBase);

In-Memory — набор концепций хранения данных, основанных на их сохранении в оперативной памяти сервера и использовании диска для хранения резервных копий. Быстродействие In-Memory баз данных по сравнению с реляционными позволит повысить отзывчивость системы, уменьшив время ожидания пользователя при выполнении сохранения и поиска терминов.

2 Конструкторская часть

2.1 Проектирование БД

2.1.1 Формализация сущностей системы

На рисунке ????? представлена диаграмма, отражающая информацию о таблицах проектируемой БД.

...Диаграмма и перечисление атрибутов (с указанием типов)...

Поле «класс» в моделях нужно для масштабирования.

2.1.2 Триггеры БД?

А они нужны? Если да, то здесь описать, какие триггеры нужны и зачем.

2.1.3 Ролевая модель

С какими таблицами (и как) могут работать разные пользователи, имеющие разные роли.

2.2 Проектирование программного комплекса

2.2.1 Архитектура приложения

Приложение состоит из нескольких элементов: пользовательский интерфейс, базы данных и внешние сервисы. Архитектура приложения определяет, как эти компоненты будут взаимодействовать друг с другом, а также устанавливает границы между разными частями приложения и их ответственностями.

2.2.1.1. Монолитная архитектура

Монолитная архитектура — это традиционная модель программного обеспечения, которая представляет собой единый модуль, работающий автономно и независимо от других приложений. Монолитная архитектура представляет собой вычислительную сеть с единой базой кода, в которой объединены все бизнес-задачи. Чтобы внести изменения в такое приложение, необходимо обновить весь стек через базу кода, а также создать и развернуть обновленную версию интерфейса, находящегося на стороне службы. Это ограничивает работу с обновлениями и требует много времени.

Монолитную архитектуру можно использовать на начальных этапах про-

ектов, чтобы облегчить развертывание и управление кодом. Это позволяет сразу выпускать все, что есть в монолитном приложении.

2.2.1.2. Микросервисная архитектура

Микросервисная архитектура представляет собой метод организации архитектуры, основанный на ряде независимо развертываемых служб. Эти службы реализуют независимую бизнес-логику и, как правило, имеют собственную базу данных. Обновление, тестирование, развертывание и масштабирование выполняются внутри каждой службы. Микросервисы разбивают крупные задачи, характерные для конкретного бизнеса, на несколько независимых кодовых баз. Микросервисы в составе приложения должны иметь независимую логику и ограниченную зону ответственности.

При переходе от монолитной архитектуры к микросервисной возникает задача организации взаимодействия компонентов приложения (в частности, транспорта данных). Соответственно, часть проблем переходит из плоскости кода на инфраструктурный и транспортный уровень.

2.2.1.3. Выбор архитектуры приложения

TODO: сравнение и подведение итогов.

Выбираем микросервисы, потому что очень сильно выигрываем от плюсов (особенно от разных языков, но пока не говорю, каких).

2.2.2 Связь компонентов приложения

2.2.2.1. REST API

2.2.2.2. gRPC

2.2.2.3. Выбор взаимодействия компонентов приложения

gRPC, потому что между сервисами передаётся много данных (структуры, представляющие очень большие тексты), поэтому критически важен быстрый транспорт.

2.2.3 Структура программного комплекса

На рисунке ????? представлена структура программного комплекса, оформленная в виде диаграммы развёртывания. Она отражает компоненты, необходи-

мые для работы системы, а также способы их взаимодействия.

Обязательно здесь упомянуть про контейнеризацию и Docker!

2.2.4 Диаграммы последовательностей

2.2.5 Паттерны проектирования

Паттерны проектирования повышают степень повторного использования проектных и архитектурных решений. Они помогают выбрать альтернативные решения, упрощающие повторное использование системы, и избежать тех альтернатив, которые его затрудняют. Паттерны улучшают качество документации и сопровождения существующих систем, поскольку они позволяют явно описать взаимодействия классов и объектов, а также причины, по которым система была построена так, а не иначе [16].

Далее будут описаны паттерны проектирования, которые будут использованы при разработке программного комплекса.

2.2.5.1. Repository

Repository (репозиторий) [17] — это слой абстракции, инкапсулирующий в себе всё, что относится к способу хранения данных. Он предназначен для отделения бизнес-логики от деталей реализации слоя доступа к данным.

Паттерн Репозиторий стал популярным благодаря DDD (Domain Driven Design). В отличие от Database Driven Design, в DDD разработка начинается с проектирования бизнес логики, причём во внимание принимаются только особенности предметной области, а всё, что связано с особенностями хранения данных, игнорируется.

Применение данного паттерна не предполагает создание только одного объекта репозитория во всем приложении. Хорошей практикой считается создание отдельных репозиториях для каждого бизнес-объекта или контекста, например: OrdersRepository, UsersRepository, AdminRepository.

Репозиторий — это высокоуровневая абстракция доступа к данным. Интерфейс каждого конкретного репозитория определяется в слое бизнес-логики наряду с классами предметной области. Реализация каждого репозитория на-

ходится в слое доступа к данным (Data Access Layer, DAL), который состоит из реализации каждого репозитория, ORM-специфичных классов, сущностей, классов-сопоставлений (mapping), контекстов данных и т.д.

2.2.5.2. Dependency injection

Dependency injection (инъекция зависимостей) [18] — это набор принципов и паттернов проектирования программного обеспечения, который позволяет разрабатывать свободно связанный код.

В программной инженерии внедрение зависимостей — это техника, при которой один объект предоставляет зависимости другому объекту. Зависимость — это объект, который может быть использован, например, в качестве сервиса. Вместо того чтобы клиент указывал, какой сервис он будет использовать, что-то указывает клиенту, какой сервис использовать. Инъекция относится к передаче зависимости (сервиса) в объект (клиент), который будет его использовать. Сервис становится частью состояния клиента. Передача сервиса клиенту, вместо того чтобы позволить клиенту создать или найти сервис, является фундаментальным требованием паттерна.

Создание объектов непосредственно в классе является негибким, поскольку фиксирует класс на определенных объектах и делает невозможным дальнейшее изменение инстанцирования независимо от класса. Это не позволяет классу быть многократно используемым, если потребуются другие объекты, и затрудняет тестирование класса, поскольку реальные объекты не могут быть заменены имитационными объектами.

Зависимость от интерфейса является более гибкой, чем зависимость от конкретных классов. Объектно-ориентированные языки предоставляют способы, с помощью которых можно заменить эти абстракции конкретными реализациями во время выполнения. Инъекция зависимостей помогает кодовую базу гибкой и пригодной для повторного использования.

2.2.5.3. Template method

Template Method (шаблонный метод) [16] — паттерн поведения классов.

Шаблонный метод определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Это позволяет подклассам переопределять отдельные шаги алгоритма, не меняя его общей структуры.

Основные условия для применения паттерна шаблонный метод:

- однократное использование инвариантных частей алгоритма, при этом реализация изменяющегося поведения остается на усмотрение подклассов;
- необходимость вычленить и локализовать в одном классе поведение, общее для всех подклассов, чтобы избежать дублирования кода;
- управление расширениями подклассов (шаблонный метод можно определить так, что он будет вызывать операции-зацепки (hooks) в определенных точках, разрешив тем самым расширение только в этих точках);

Шаблонные методы — один из фундаментальных приемов повторного использования кода. Они играют особенно важную роль в библиотеках классов, поскольку предоставляют возможность вынести общее поведение в библиотечные классы. Шаблонные методы приводят к инвертированной структуре кода, которая подразумевает, что родительский класс вызывает операции подкласса, а не наоборот.

3 Технологическая часть

3.1 Анализ СУБД

Взять в рассмотрение хотя бы 3 из них:

3.1.1 SQLite?

3.1.2 MySQL

3.1.3 Oracle

3.1.4 Microsoft SQL Server

3.1.5 PostgreSQL

3.1.6 MongoDB

3.1.7 Redis

3.1.8 Выбор СУБД для решения задачи

PostgreSQL — основное хранилище, Redis — кеш данных текущей сессии (по окончании сессии перекидывается на диск на стороне сервера (после возврата из функции обработки подключения, т.е. на стороне сервера надо разделять (идентифицировать) конкретные сессии (не пользователей!))).

3.2 Средства реализации

Что на чём написано, как, куда и с помощью чего деплоилось (упоминуть Docker и контейнеризацию). Если в конечном варианте останется Envoy, то рассказать, зачем он нужен.

Golang + Python + SolidJS/React

Внешние средства для мониторинга состояния системы: Docker Desktop (или UI сервиса, на котором развёрнуто приложение), Grafana, PGAdmin.

3.3 Детали реализации

3.3.1 Триггеры БД?

3.3.2 Роли БД

3.3.3 Примеры работы ПО

4 Экспериментально-исследовательская часть

4.1 Цель проводимых измерений

4.2 Описание проводимых измерений

Измеряем время операций сохранения терминов в БД и поиска терминов по тегам (INSERT и SELECT) на примере нескольких текстов с возрастающим количеством терминов (желательно кратным 10^n).

4.3 Инструменты измерения времени работы программы

4.4 Результаты проведённых измерений

Заключение

В рамках научно-исследовательской работы была проведена классификация сетевых операционных систем для устройств интернета вещей.

В результате сравнения были выделены:

- Azure Sphere, Windows 10 IoT и Amazon FreeRTOS как наиболее функциональные и масштабируемые;
- ОСРВ МАКС и KasperskyOS как наиболее доступные с точки зрения использования прикладных служб;
- Ubuntu Core и Raspbian как наиболее адаптированные для бытового применения.

В ходе выполнения данной работы были решены следующие задачи.

1. Проведён анализ предметной области интернета вещей;
2. Рассмотрены существующие операционные системы для устройств интернета вещей;
3. Сформулированы критерии сравнения и оценки рассмотренных операционных систем;
4. Проведён сравнительный анализ существующих решений по выделенным критериям.

Таким образом, все поставленные задачи были выполнены, поставленная цель достигнута.

Список использованных источников

1. Когаловский М.Р. Энциклопедия технологий баз данных. – Москва: Финансы и статистика, 2002. – 800 с.
2. Алферова Т.К., Леонов А.В., Филиппов К.В. О состоянии автоматизированной базы данных терминов и определений в области ОП // Компетентность. – 2014. – № 7(118). – С. 10-14.
3. Горбач Т.А., Грибова В.В., Окунь Д.Б., Петряева М.В., Шалфеева Е.А., Шахгельдян К.И. База терминов нейрохирургии для интеллектуальной обработки биомедицинских данных // Сборник материалов XIII международной научной конференции: «Системный анализ в медицине». – Благовещенск, 2019. – С. 82-85.
4. Кузнецов И.О. Автоматическое извлечение двусловных терминов по тематике «Нанотехнологии в медицине» на основе корпусных данных // Научно-техническая информация. Сер. 2. – 2013. – № 5. – С. 25-33.
5. Becerro F. B. Phraseological variations in medical-pharmaceutical terminology and its applications for English and German into Spanish translations // SciMedicine Journal. – 2020. – № 2(1). – P.22-29. DOI: 10.28991/SciMedJ-2020-0201-4.
6. Simon N. I., Kešelj V. August. Automatic term extraction in technical domain using part-of-speech and common-word features // Proceedings of the ACM Symposium on Document Engineering. – 2018. – P. 1-4. DOI:10.1145/3209280.3229100.
7. Клышинский Э.С., Кочеткова Н.А., Карпик О.В. Метод выделения коллокаций с использованием степенного показателя в распределении Ципфа // Новые информационные технологии в автоматизированных системах. – 2018. – № 21. – С. 220-225.

8. Кочеткова Н.А. Метод извлечения технических терминов с использованием усовершенствованной меры странности // Научно-техническая информация. Сер. 2. – 2015. – № 5. – С. 25-32;
Kochetkova N.A. A Method for Extracting Technical Terms Using the Modified Weirdness Measure // Automatic Documentation and Mathematical Linguistics. – 2015 – Vol. 49, № 3. – P. 89-95.
9. Захаров В.П., Хохлова М.В. Автоматическое извлечение терминов из специальных текстов с использованием дистрибутивно-статистического метода как инструмент создания тезаурусов // Структурная и прикладная лингвистика. – 2012. – № 9. – С. 222-233.
10. Terry A., Hoste V., Lefever E. In no uncertain terms: a dataset for monolingual and multilingual automatic term extraction from comparable corpora // Language Resources and Evaluation. – 2020. – Vol. 54, № 2. – P. 385-418. DOI:10.1007/s10579-019-09453-9.
11. Бутенко Ю.И. Строганов Ю.В., Сапожков А.М. Метод извлечения русскоязычных многокомпонентных терминов в корпусе научно-технических текстов // Прикладная информатика. – 2021. – № 6. – С. 21-27. DOI: 10.37791/2687-0649-2021-16-6-21-27.
12. Бутенко Ю.И. Строганов Ю.В., Сапожков А.М. Система извлечения многокомпонентных терминов и их переводных эквивалентов из параллельных научно-технических текстов // НТИ. Сер. 2. ИНФОРМ. ПРОЦЕССЫ И СИСТЕМЫ. – 2022. – № 9. – С. 12-21. ISSN 0548-0027.
13. К. Дж. Дейт SQL и реляционная теория. Как грамотно писать код на SQL. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 480 с., ил. ISBN 978-5-93286-173-8

14. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс 2005. — 1328 с.: ил. — Парал. тит. англ. ISBN 5-8459-0788-8 (рус.)
15. Маркин, А. В. Системы графовых баз данных. Neo4j : учебное пособие для вузов / А. В. Маркин. — Москва : Издательство Юрайт, 2021. — 178 с. — (Высшее образование). ISBN 978-5-534-13996-9
16. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. — СПб.: Питер, 2020. — 448 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-1595-2
17. The Repository Pattern Explained [Электронный ресурс]. — Режим доступа: <https://blog.sapiensworks.com/post/2014/06/02/The-Repository-Pattern-For-Dummies.aspx> (дата обращения: 06.02.2023).
18. Seemann, M. and van Deursen, S. Dependency Injection. Principles, Practices, and Patterns. — Manning, 2019. — 552 с. ISBN 978-1-6172-9473-0

Приложение А

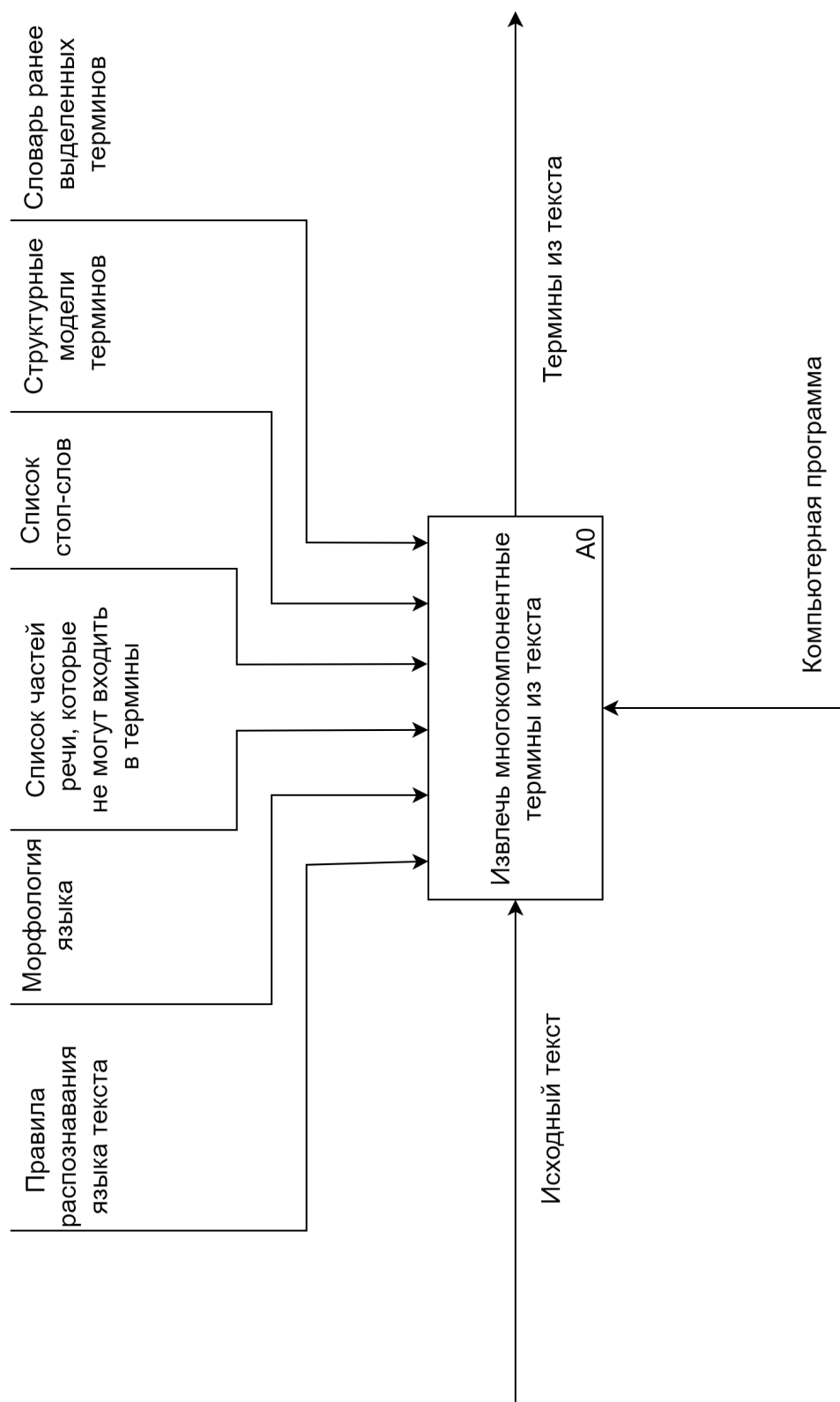


Рисунок 3 – Функциональная схема работы системы извлечения многокомпонентных терминов, верхний уровень

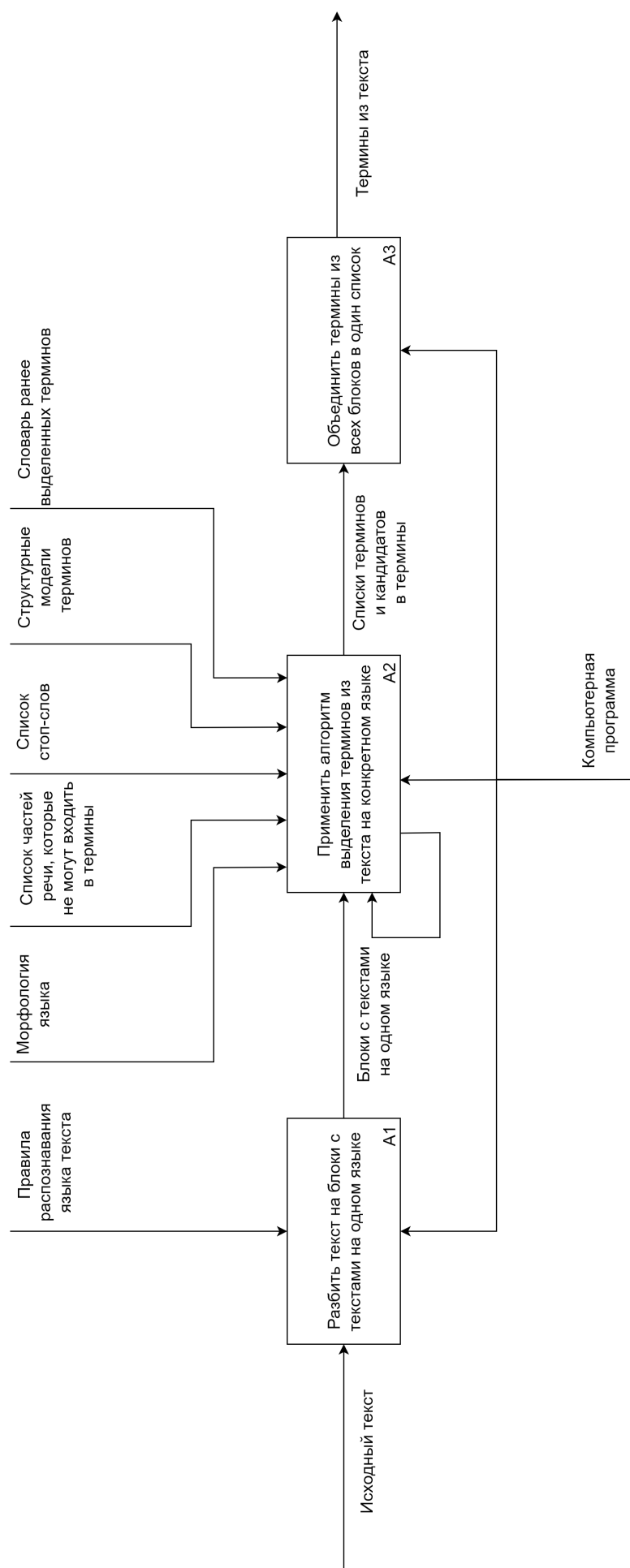


Рисунок 4 – Функциональная схема работы системы извлечения многокомпонентных терминов, декомпозиция уровня

A0

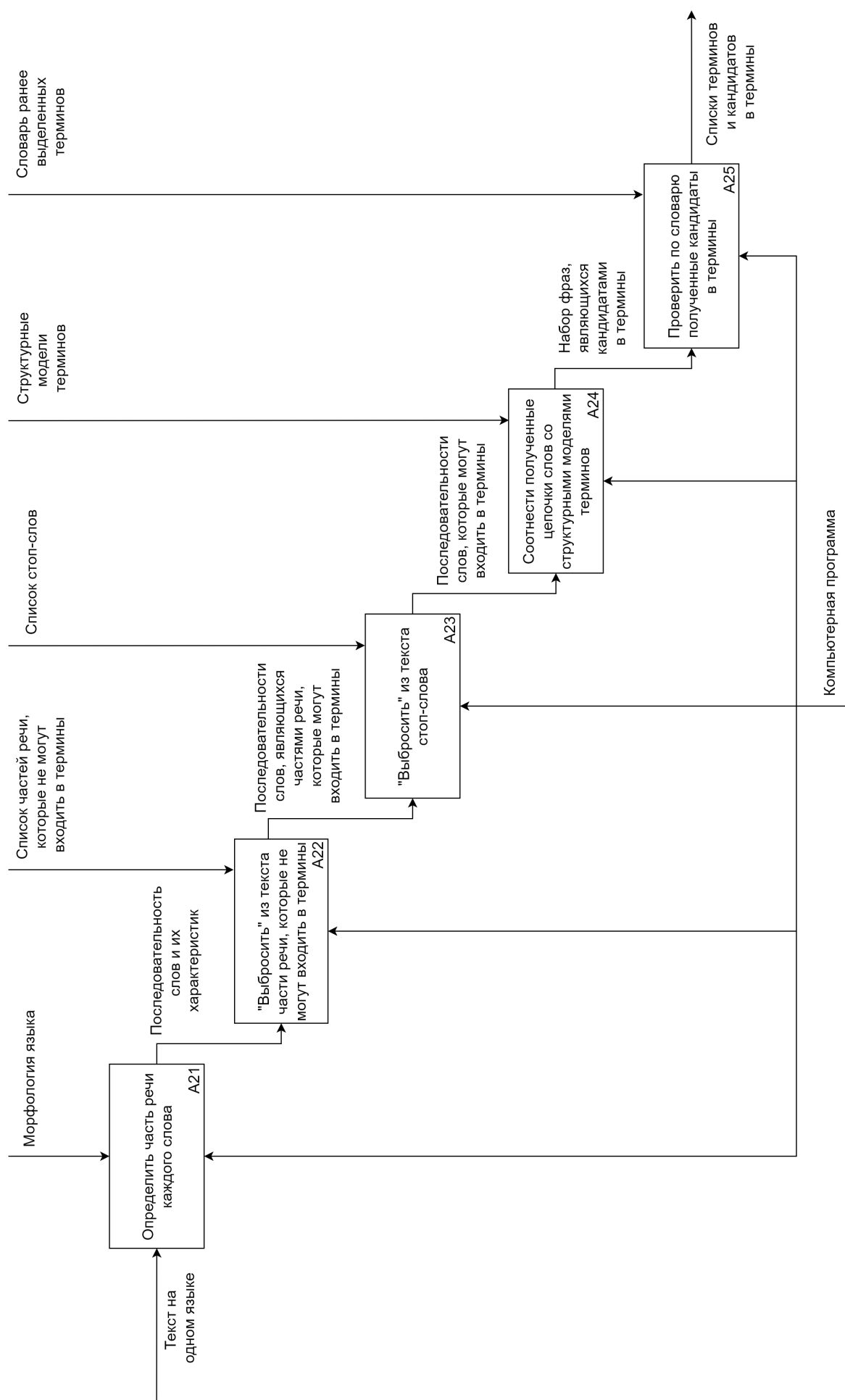


Рисунок 5 – Функциональная схема работы системы извлечения многокомпонентных терминов, декомпозиция уровня

Приложение Б

Презентация.