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

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка системы извлечения многокомпонентных терминов и их переводных эквивалентов из параллельных научно-технических текстов»

Студент **ИУ7-63Б** _____ **Сапожков А. М.**

Руководитель _____ **Строганов Ю. В.**

2023 г.

Реферат

Расчетно-пояснительная записка 65 с., 13 рис., 2 табл., 35 ист., 7 прил.

Ключевые слова: научно-технические тексты, многокомпонентный термин, извлечение терминов, базы данных, PostgreSQL, NoSQL, Redis, InfluxDB, REST API, кеширование 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

Определения, обозначения и сокращения

Единица языка — элемент системы языка, неразложимый в рамках определённого уровня членения текста и противопоставленный другим единицам в подсистеме языка, соответствующей этому уровню.

Терминологическая единица — элемент терминологической системы, которая является языковым выражением системы понятий определенной предметной области.

Термин — слово или словосочетание, являющееся названием определённого понятия в определенной предметной области.

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.

Бизнес-сценарий — сценарий взаимодействия пользователя с программным продуктом для достижения конкретной цели.

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

Компонент — единица развёртывания ПО, которая является реализацией сервиса, содержащей 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 материла – 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 базе данных.
   5.1. Если термин-кандидат есть в базе данных, то он извлекается как термин.
   5.2. Если полученный термин-кандидат отсутствует в базе данных, то он от-

правляется терминологу для ручной обработки.

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

1.2 Требования к приложению

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 данных

Разрабатываемая база данных должна хранить информацию о следующих сущностях:

— пользователи;
— русскоязычные термины;
— иностранные термины;
— характеристики терминов;
— структурные модели терминов;
— части речи;
— контексты употребления терминов.

На рисунке 1 представлена ER-диаграмма сущностей в нотации Чена, описывающая сущности, их атрибуты и связи между сущностями в разрабатывае-

мой базе данных.

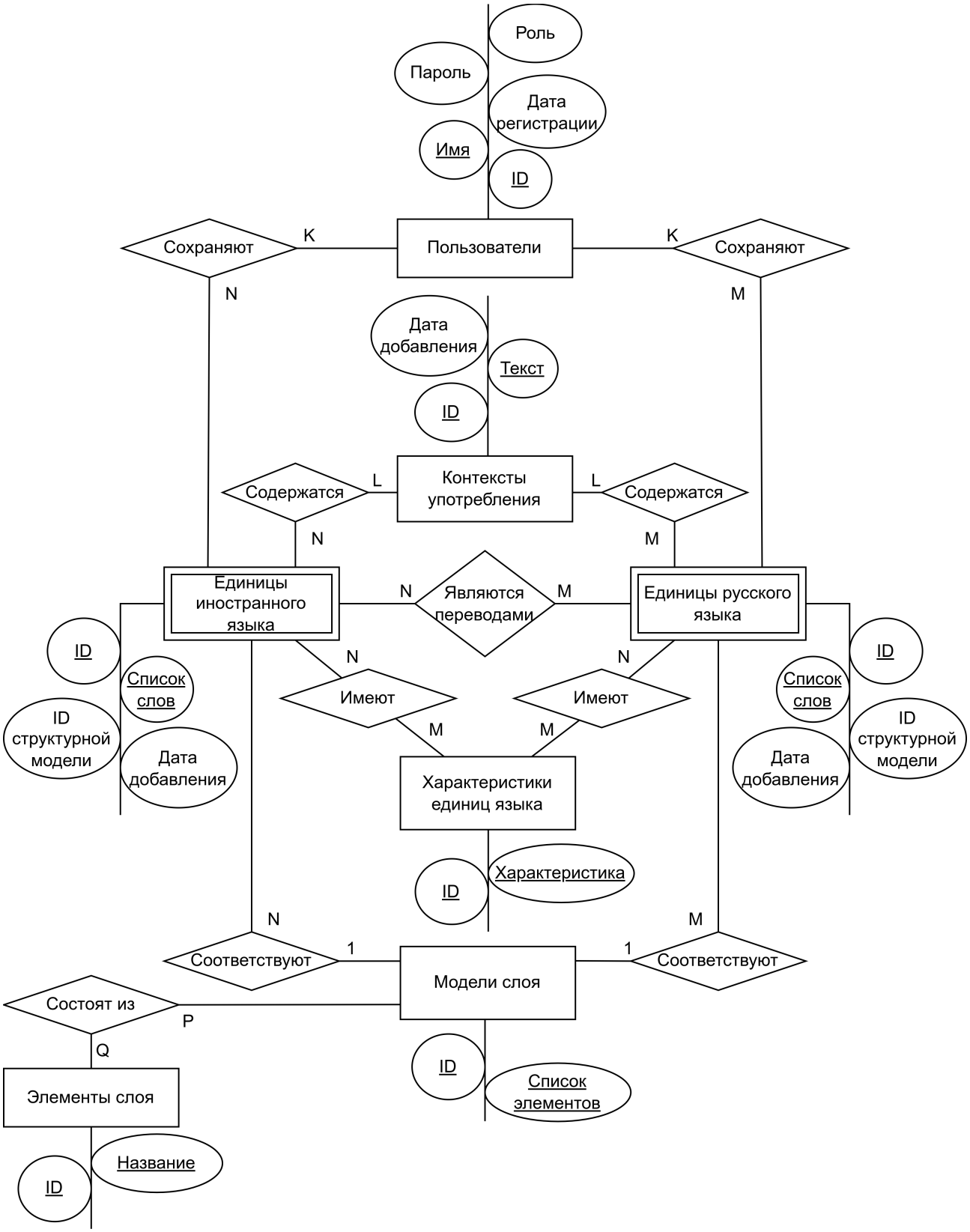

Рисунок 1 – ER-диаграмма сущностей базы данных

Стоит отметить, что тексты могут размечаться на нескольких слоях. Это означает, что текст может разбиваться на разные единицы языка и из него можно выделять отдельные слова, термины, синтаксические деревья, семантические падежи и т.д. Соответственно, разрабатываемая база данных должна иметь многослойную структуру: работа с одними и теми же сущностями должна производиться по отдельности для каждого слоя разметки текстов.

1.4 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», и имеющий возможность анализировать и редактировать все термины, хранящиеся в базе данных. Таже данной роли доступна функция добавления нового слоя разметки текстов.
3. Администратор — пользователь, имеющий возможности, доступные роли «Преподаватель», а также имеющий в распоряжении специальные функции 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 2 представлена диаграмма вариантов использования системы в соответствии с выделенными типами пользователей.

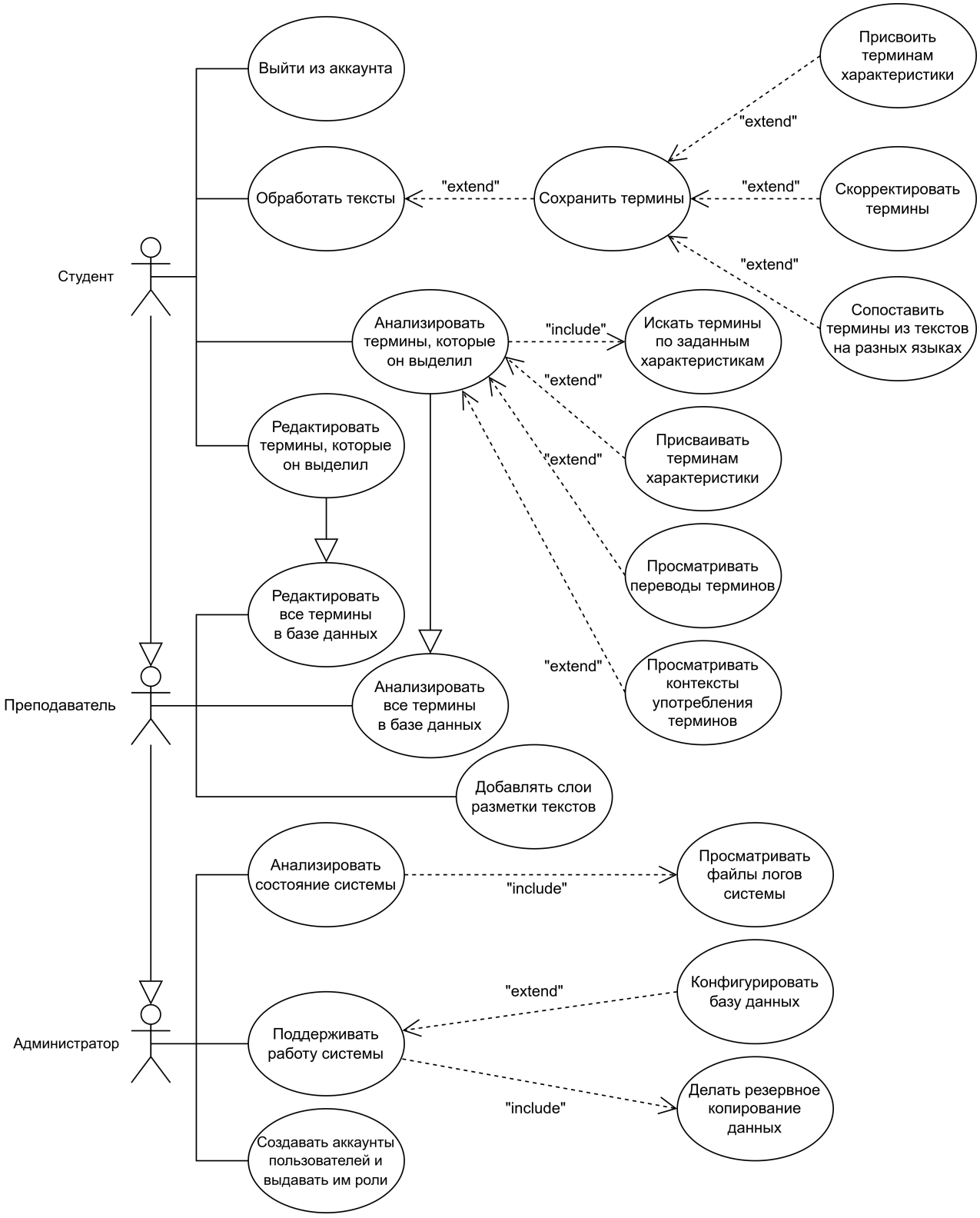

Рисунок 2 – Диаграмма вариантов использования

1.5 Анализ моделей данных

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 (hierarchic) и сетевые (network) [14].

1.1. **Инвертированные списки (файлы)**.

База данных 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 базы данных задаются теми программами, которые с ней работают. Одно из немногих ограничений, которое может присутствовать — это ограничение, задаваемое уникальным индексом.

1.2. **Иерархические**.

Иерархическая модель данных состоит из объектов с указателями от родительских объектов к дочерним, соединяя вместе связанную информацию. Иерархические базы данных 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 данных относятся элемент (узел)

и связь. Узел — это совокупность атрибутов данных, описывающих некоторый объект. Сетевые базы данных 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**.

Определение реляционной системы требует [14], 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.

Реляционные базы данных основаны на информационном принципе: 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 следующих компонентов [14].

— Неограниченный набор скалярных типов (включая, в частности, логический тип).
— Генератор типов отношений и соответствующая интерпретация для сгенерирован ных 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 постреляционных СУБД 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 данных

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 вторичной памяти 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.

1.6 Вывод из аналитической части

В данном разделе были описан метод извлечения многокомпонентных терминов из научно-технических текстов, на основе которого были сформулированы требования к приложению. Также были описаны сущности базы данных и определены следующие роли пользователей: студент, преподаватель и администратор. Были проанализированы модели данных: для долговременного хранения данных будет использоваться реляционная модель, а для кеширования данных пользовательских сессий — нереляционная (In-Memory хранилище).

2 Конструкторская часть

В данном разделе будет формализована структура базы данных и будут описаны подходы к разработке приложения.

2.1 Проектирование базы данных

2.1.1 Формализация сущностей системы

На рисунке 3 представлена диаграмма, отражающая информацию о таблицах проектируемой базы данных.

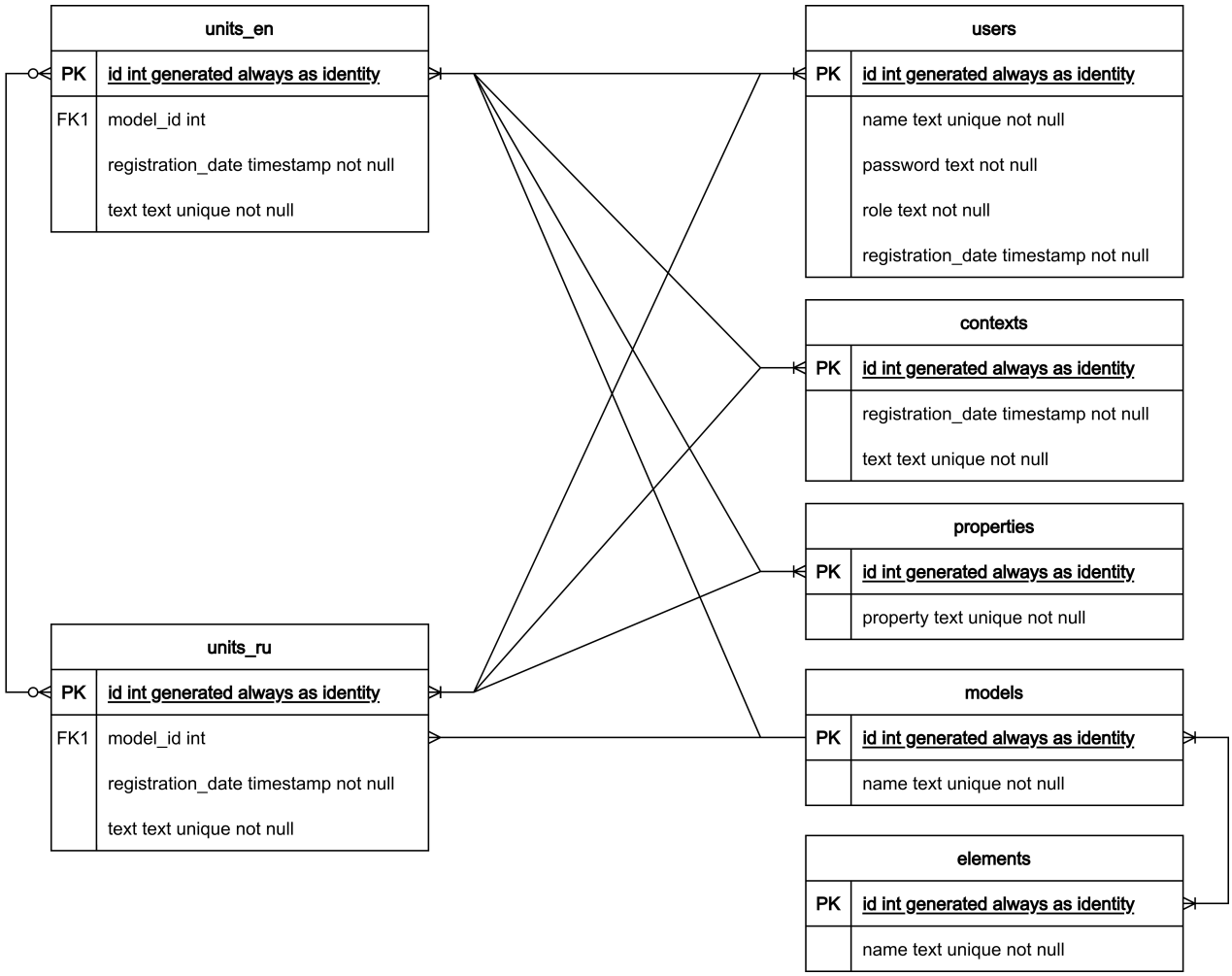

Рисунок 3 – Диаграмма базы данных

Для обеспечения возможности разметки текстов на нескольких слоях необходимо определить, какие таблицы должны создаваться отдельно для каждого слоя. Таблицы users, contexts и properties не должны зависеть от слоя разметки текстов, а все остальные таблицы должны хранить информацию о каждом слое

отдельно.

2.1.2 Ролевая модель

Для поддержания целостности данных необходимо задать ограничения на операции доступа к данным. В таблице 1 представлено описание прав пользователей на работу с таблицами разрабатываемой базы данных.

Таблица 1 – Роли базы данных

Роль	Права на таблицы			
	Выборка	Вставка	Обновление	Удаление
Студент	contexts, properties, units_ru, units_en, models, elements	contexts, properties, units_ru, units_en	contexts, properties, units_ru, units_en	properties
Преподаватель	contexts, properties, units_ru, units_en, models, elements	contexts, properties, units_ru, units_en, models, elements	contexts, properties, units_ru, units_en, models, elements	contexts, properties, models, elements
Администратор	users, contexts, properties, units_ru, units_en, models, elements	users, contexts, properties, units_ru, units_en, models, elements	users, contexts, properties, units_ru, units_en, models, elements	users, contexts, properties, units_ru, units_en, models, elements

Если пользователь имеет право на работу с какой-либо таблицей базы данных, то он также может работать со всеми соответствующими таблицами-связками.

2.1.3 Хранимая процедура базы данных

Для создания нового слоя разметки текстов необходимо совершить последовательность действий над несколькими таблицами базы данных, которые можно объединить в хранимую процедуру. Её задачей будет создание связанных таблиц для хранения данных о новом слое, а также выдача пользователям прав на работу с ними.

На рисунке 4 представлена схема алгоритма работы хранимой процедуры, добавляющей новый слой разметки текстов.

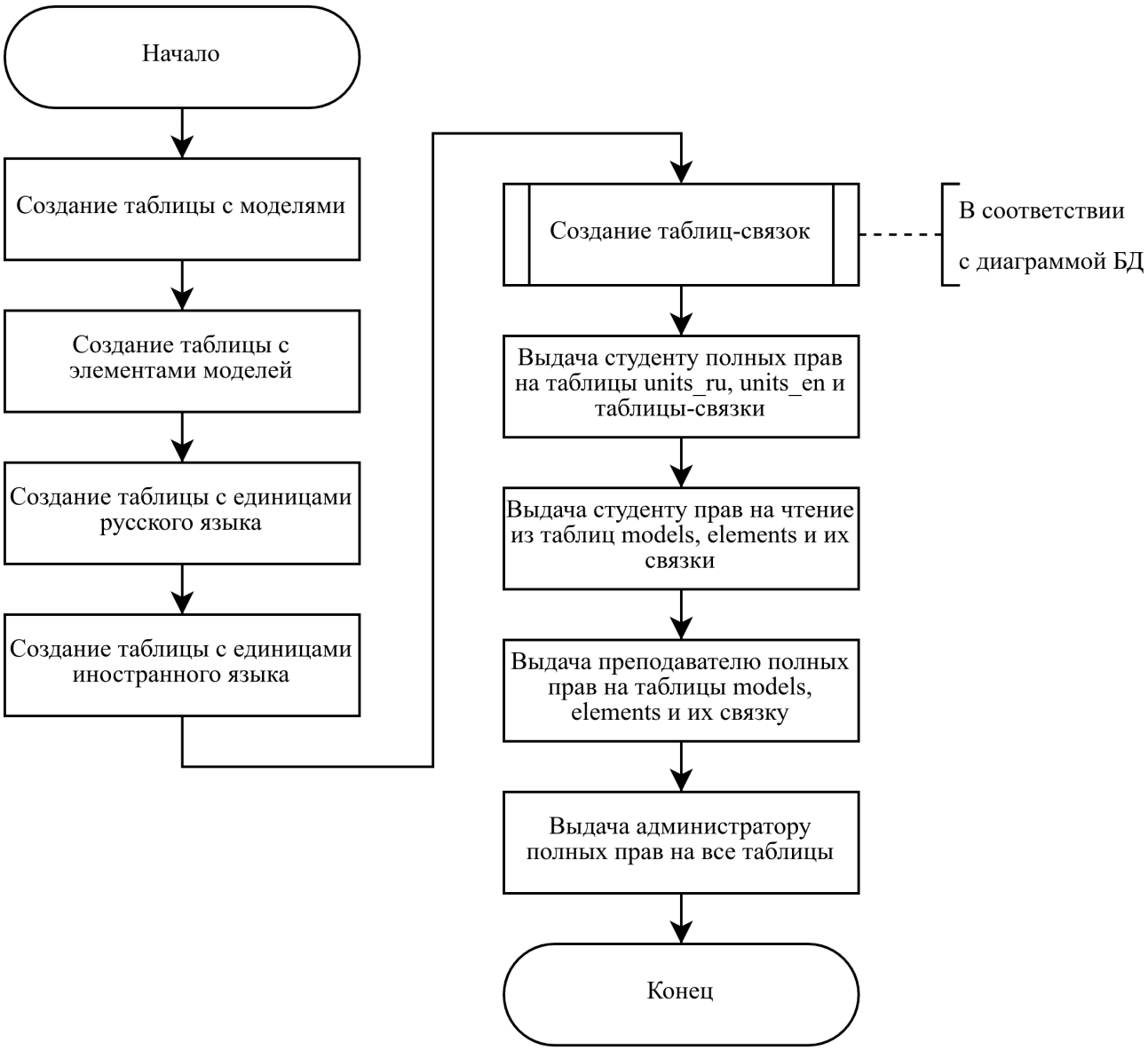

Рисунок 4 – Схема алгоритма добавления нового слоя разметки текстов

2.2 Проектирование программного комплекса

2.2.1 Бизнес-сценарии

Для разработки функций приложения необходимо описать его бизнес-логику. Диаграммы в нотации BPMN, отражающие формализацию бизнес-правил, представлены в приложении Б.

2.2.2 Архитектура приложения

Приложение состоит из нескольких компонентов: 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.

2.2.2.1. Монолитная архитектура

Монолитная архитектура [16] — 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 проектов, чтобы облегчить развертывание и управление кодом. Это позволяет сразу выпускать всё, что есть в монолитном приложении.

2.2.2.2. Микросервисная архитектура

Микросервисная архитектура [16] 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 масштабирова-

ние 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.

2.2.2.3. Выбор архитектуры приложения

Основные преимущества использования монолитной архитектуры:

1) начальная разработка;
2) передача данных между компонентами системы;
3) единая кодовая база;
4) хранение состояния (stateful);
5) инфраструктура развертывания;
6) единое версионирование проекта;
7) производительность;
8) интеграционное тестирование.

Основные преимущества использования микросервисной архитектуры:

1) независимая разработка и выпуск;
2) применение разных технологий для каждой выделенной задачи;
3) независимое развёртывание и горизонтальное масштабирование;
4) модульное тестирование;
5) отказоустойчивость системы;
6) повторное использование кода;
7) возможность полностью переписать отдельные компоненты приложения.

Для решения поставленной задачи будет использоваться микросервисная архитектура, так как она позволяет независимо разрабатывать и масштабировать компоненты приложения.

2.2.3 Связь компонентов приложения

Программные компоненты приложения могут взаимодействовать друг с другом с помощью интерфейсов прикладного программирования (Application Programming Interface, API). API описывает, как выполнять клиентские запросы, какие структуры данных использовать и каких стандартов должны придерживаться клиенты. В нем также описываются виды запросов, которые один компонент может отправлять другому. Для разработки API широко используются архитектурные стили REST API и gRPC.

2.2.3.1. REST API

REST, representational state transfer (англ. передача репрезентативного состояния) [19] описывает архитектуру клиент-сервер, в которой данные сервера становятся доступными для клиентов через формат обмена сообщениями JSON или XML.

REST определяется следующими архитектурными ограничениями [19].

1. Модель клиент-сервер.
2. Отсутствие хранения состояния клиента на сервере.
3. Кеширование ответов сервера на стороне клиента.
4. Унифицированный интерфейс.
5. Многоуровневая система.
6. Код по запросу (необязательное ограничение).

Приложение, соответствующее данным архитектурным ограничениям, квалифицируется как «RESTful». Это сеть веб-страниц (виртуальная машина), по которой пользователь перемещается с помощью ссылок (переходы состояний) и в результате попадает на нужную страницу (демонстрация состояния приложения).

Также важно отметить, что REST API практически всегда использует протокол HTTP. Это наиболее распространенный формат, используемый для разработки веб-приложений или соединения микросервисов. Если веб-приложение

реализует REST API, то клиенты могут использовать каждый его компонент в качестве ресурса. Обычно ресурсы доступны через общий интерфейс, который реализует различные HTTP-методы, такие как GET, POST, DELETE и PUT.

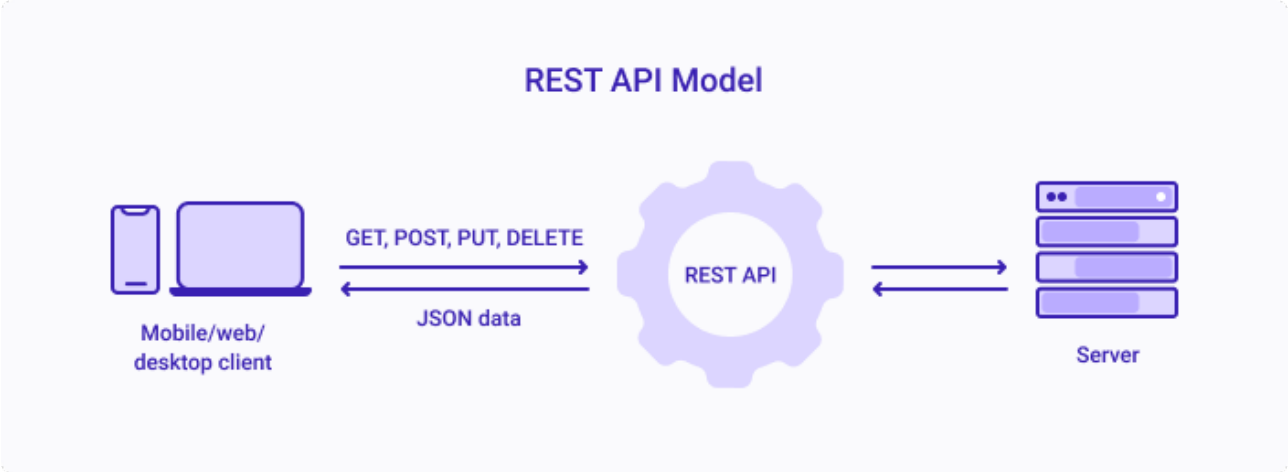

Рисунок 5 – Модель REST API

В RESTful API пользователь отправляет запрос на URL-адрес — унифицированный указатель ресурсов, который вызывает ответ с полезной нагрузкой в JSON, XML или любой другой поддерживаемый формат данных. Полезная нагрузка представляет собой ресурс, который нужен пользователю. Общие запросы клиентов включают

— HTTP-метод, указывающий, что должно обрабатываться на ресурсе;

— путь к ресурсу;

— заголовок с данными о запросе;

— полезная нагрузка сообщения для конкретного клиента.

2.2.3.2. gRPC

gRPC, remote procedure call (англ. удалённый вызов процедур) [17] — это протокол RPC, реализованный поверх HTTP/2 (протокол прикладного уровня). Основные особенности gRPC:

— бинарный протокол (HTTP/2);

— мультиплексирование множества запросов на одно соединение (HTTP/2);

— сжатие заголовков (HTTP/2);

— строго типизированный сервис и определение сообщения (Protobuf);
— идиоматические реализации библиотек клиент/сервер на многих языках (Golang, C++, Python и другие);
— интеграция с такими компонентами экосистемы, как обнаружение сервисов, преобразователь имен, балансировщик нагрузки, трассировка и мониторинг и другие.

Удаленный вызов процедур — это веб-архитектура, позволяющая выполнять запросы на сервере с использованием предопределенных форматов сообщений. При этом сервер может быть как локальным, так и удалённым.

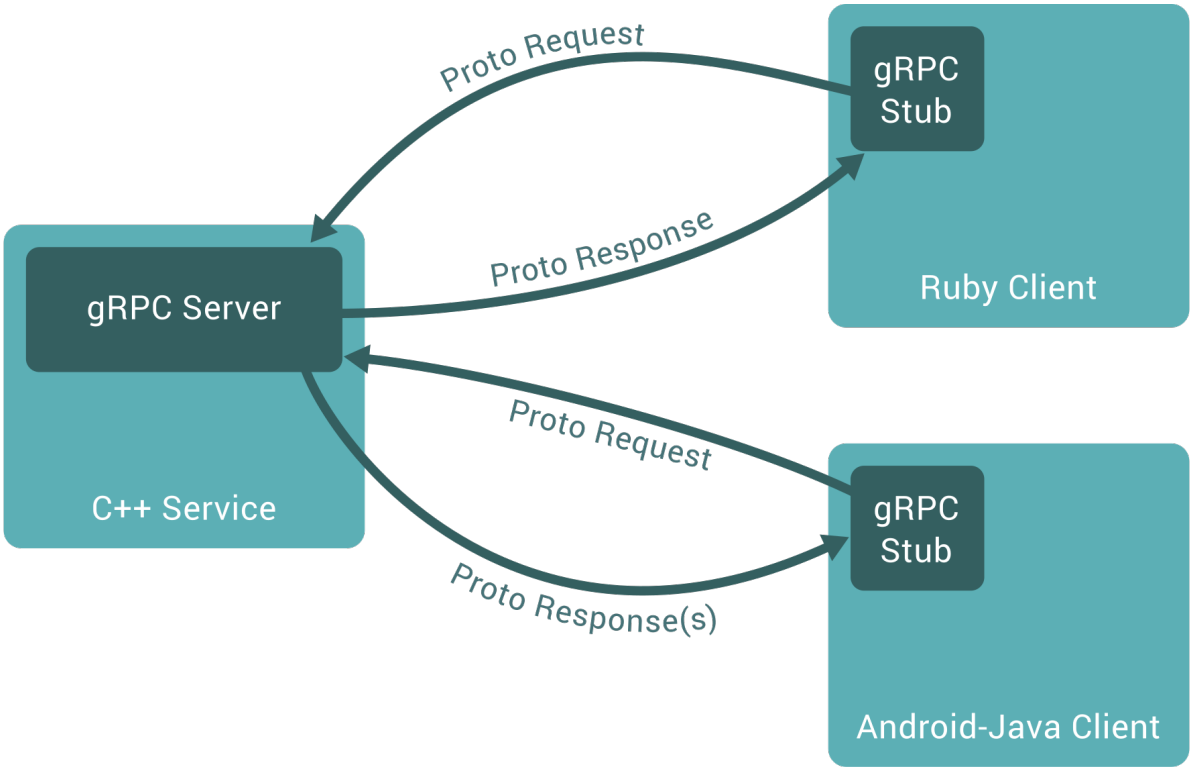

Рисунок 6 – Модель gRPC

Эта архитектура API позволяет нескольким функциям, созданным на разных языках программирования, работать вместе благодаря API. gRPC использует формат обмена сообщениями Protobuf (англ. буферы протокола), который используется для сериализации структурированных данных. Для определенного круга задач API gRPC является более эффективной альтернативой REST API.

2.2.3.3. Выбор способа взаимодействия компонентов приложения

В таблице 2 представлено сравнение gRPC и REST API.

Таблица 2 – Сравнительная характеристика gRPC и REST API

Характеристика	gRPC	REST API
Формат коммуникации	Унарные двусторонние запросы или потоковая передача	Только клиентские запросы
Способ коммуникации	RPC	HTTP-методы
Протокол	HTTP/2	HTTP 1.1
Формат сообщений	Protobuf (protocol buffers)	JSON/XML
Генерация кода	С помощью компилятора Protobuf	Сторонние решения, такие как Swagger [18]

Для решения поставленной задачи будет использоваться REST API по следующим причинам:

1. Универсальность: позволяет связывать любые сервисы, которые могут принимать или отправлять HTTP-запросы.
2. Скорость развёртывания.
3. Поддержка инструментов описания API (Swagger, ReDoc, Apiary и другие).

2.2.4 Структура программного комплекса

На рисунке 7 представлена структура программного комплекса, оформленная в виде диаграммы развёртывания. Она отражает компоненты системы и способы их взаимодействия.

Для упрощения процесса разработки и развёртывания ПО предлагается использовать контейнеризацию, чтобы изолировать компоненты приложения друг от друга. В отличии от виртуальных машин, имеющих ОС хоста и гостевую ОС, контейнеры размещаются на одном физическом сервере с ОС хоста, которая разделяет их между собой. Совместное использование ОС хоста сразу

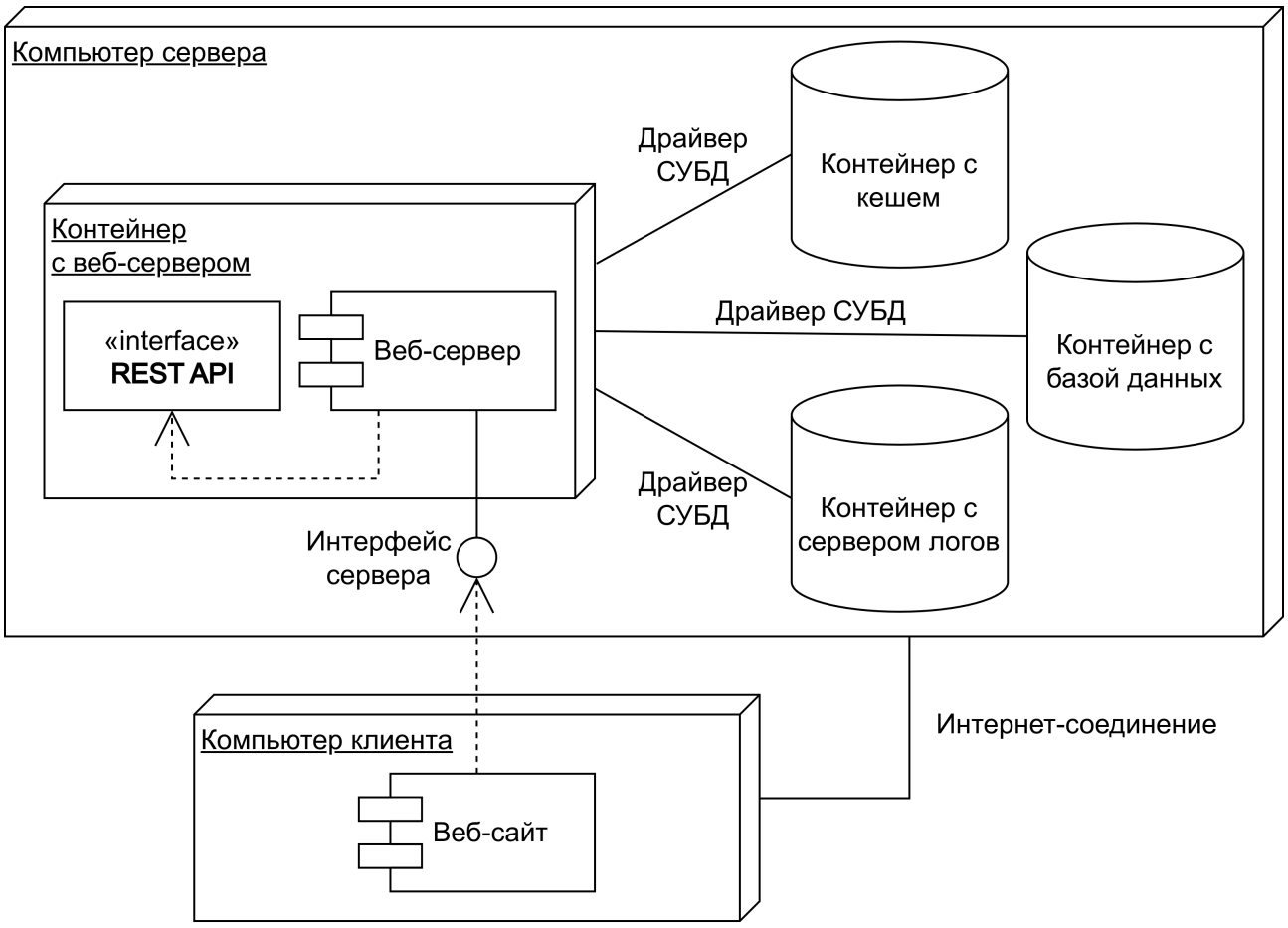

Рисунок 7 – Диаграмма развёртывания программного комплекса

несколькими контейнерами делает их менее требовательными к аппаратному обеспечению.

Также в целях повышения безопасности системы предлагается использовать выделенный сервер логов. Для организации базы данных для этого сервера можно использовать СУБД, ориентированную на обработку временных рядов[1] (СУБД-ВР) [20]. Такой формат хранения данных позволит в динамике отслеживать состояние системы при минимальных затратах ресурсов на хранение и обработку событий.

2.2.5 Паттерны проектирования

Паттерны проектирования повышают степень повторного использования проектных и архитектурных решений. Они помогают выбрать альтернативные

[1]Временной ряд (time series) — последовательность хронологически упорядоченных числовых значений, отражающих течение некоторого процесса или явления.

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 [21].

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

2.2.5.1. Repository

Repository (репозиторий) [22] — 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 репозиториев 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 находится 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) [23] — это набор принципов и паттернов проектирования программного обеспечения, который позволя-

ет 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) [21] — 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.

2.3 Вывод из конструкторской части

В данном разделе была проведена формализация сущностей и ролей системы, разработана хранимая процедура для создания нового слоя разметки текстов. Также были формализованы основные бизнес-правила и выбраны подходы к разработке приложения. Развёртывание приложения будет осуществляться с помощью контейнеров. Для хранения логов системы будет использоваться СУБД временных рядов (СУБД-ВР).

3 Технологическая часть

В данном разделе будет описан выбор технологий для разработки и развёртывания базы данных и приложения, а также будут приведены детали реализации ПО.

3.1 Выбор СУБД

3.1.1 Долговременное хранение данных

В качестве основного хранилища данных была выбрана объектно-реляционная СУБД PostgreSQL [24], которая позволит реализовать многослойную структуру хранения данных. Также стоит отметить, что PostgreSQL сертифицирована ФСТЭК России [25] и имеет открытый исходный код.

3.1.2 Кеширование данных

Для хранения сессий было выбрано in-memory хранилище Redis [26]. Данная СУБД удовлетворяет необходимым требованиям (хранилище типа ключ-значение в оперативной памяти) и не требует дополнительной настройки для начала использования.

3.1.3 Хранение логов системы

В качестве СУБД для хранения логов. Для данной цели была выбрана СУБД-ВР InfluxDB [27]. В отличие от PostgreSQL, InfluxDB более компактно хранит данные и оптимизирована на запись данных, что подходит для организации сервера логов.

В InfluxDB данные представляются в виде двумерной таблицы [20], называемой измерением (measurement). В измерении имеется столбец с метками времени (timestamp). Остальные столбцы измерения могут принадлежать одной из двух категорий: поле или тег. Поле (field) хранит данные временного ряда и состоит из ключей (field keys) и значений (field values). Тег (tag) представляет собой метаданные поля и состоит из ключей тегов (tag key) и значений тегов (tag values). Поля не индексируются, но для тегов могут быть созданы индексы. В InfluxDB отсутствует явная схема базы данных.

В InfluxDB поддерживаются понятия серии и точки [20]. Серия (series) представляет собой набор данных, имеющих общие измерение, набор тегов и ключи полей. Точка (point) представляет собой элемент данных, состоящий из следующих компонентов: измерение, набор тегов, набор полей, метка времени. Точка однозначно идентифицируется по ее серии и метке времени. При добавлении точек данных метки времени добавляются автоматически.

Хранение данных на физическом уровне в InfluxDB основано на использовании древовидной структуры данных LSM (Log-structured merge-tree) [28], которая используется в реляционных СУБД и обеспечивает быстрый доступ к данным в случае сценария работы, предполагающего частые запросы на вставку данных. В InfluxDB также поддерживается автоматическое сжатие данных для минимимизации объема хранимых данных. [20]

В InfluxDB используется SQL-подобный язык запросов InfluxQL [29], который поддерживает непрерывные запросы (continous query) — запросы, которые запускаются автоматически с заданной периодичностью.

3.2 Средства реализации

Система, взаимодействующая с базой данных, представляет собой веб-сервер, доступ к которому осуществляется с помощью REST API. Для реализации был выбран язык программирования Golang [30], созданный для разработки микросервисных веб-приложений. Далее будет описана разработка микросервиса, предназначенного для хранения данных.

Для взаимодействия с базами данных будут использоваться драйвера, написанные для языка Golang, предоставляющие интерфейс взаимодействия посредством языка программирования.

Для реализации REST API будет использоваться веб фреймворк gin [31]. Документирование REST API будет осуществляться с помощью Swagger [18], который поддерживает протокол openAPI [32].

Для развёртывания приложения был выбран оркестратор Docker контейнеров Docker Compose [34]. Docker контейнер позволяет изолировать прило-

жение и разворачивать его на любой машине, независимо от окружения. Это реализуется благодаря инкапсуляции всех требуемых зависимостей внутри контейнера. Docker Compose связывает контейнеры в единую систему и позволяет запускать приложение, состоящее из Docker контейнеров. В отдельных контейнерах будут развёртываться следующие службы.

1. Микросервис приложения, осуществляющего доступ к данным.
2. PostgreSQL для долговременного хранения данных.
3. Redis для кеширования данных пользовательских сессий.
4. InfluxDB для хранения логов системы.
5. Swagger для документирования REST API.
6. PgAdmin для администрирования PostgreSQL.

3.3 Детали реализации

3.3.1 Роли базы данных

В конструкторской части была разработана ролевая модель, в которой выделены роли студента, преподавателя и администратора. Сценарий создания ролей и выделения им прав, соответствующих описанной ролевой модели, представлен в приложении В.

3.3.2 Хранимая процедура базы данных

В конструкторской части была разработана хранимая процедура, осуществляющая добавление нового слоя разметки текстов. PL/pgSQL [35] — процедурного расширения языка SQL, используемого в СУБД PostgreSQL. Код функции представлен в приложении Г.

3.3.3 Интерфейс приложения [TODO]

Разрабатываемая система является серверным приложением, которое принимает запросы клиентов, обрабатывает их и возвращает ответ. Клиенты могут взаимодействовать с сервером через интерфейс REST API, который обеспечивает доступ к разрабатываемой базе данных.

Описание REST API, соответствующего требованиям к проектируемому приложению, представлено в таблице ???????.

Описание REST API реализуемого приложения

3.4 Сборка и развёртывание приложения

Для сборки частей системы использовались Docker контейнеры, конфигурации которых представлены в приложении Д. Для их развертывания использовался Docker Compose. Листинг конфигурации в формате YAML представлен приложении Е.

3.5 Примеры работы ПО [TODO]

Картинки.

3.6 Вывод из технологической части

В данном разделе были приведены детали реализации ПО, использующего следующие технологии.

1. Golang — язык для написания серверной части приложения.
2. PostgreSQL — основное хранилище данных.
3. PgAdmin — инструмент для администрирования PostgreSQL.
4. PL/pgSQL — расширение языка SQL, использовавшееся для написания хранимых процедур базы данных.
5. Redis — кеш данных пользовательских сессий.
6. InfluxDB — сервер логов.
7. Swagger — инструмент документирования API сервера.
8. Docker — платформа для для автоматизации развёртывания и управления приложениями.
9. Docker Compose — оркестратор контейнеров.

4 Экспериментально-исследовательская часть [TODO]

В данном разделе….

4.1 Цель проводимых измерений

Исследовать влияние использования кеширования данных на время обработки запросов к приложению.

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).

Или измеряем RPS сервера.

4.3 Инструменты измерения времени обработки запросов

4.4 Результаты проведённых измерений

Графики.

4.5 Вывод из экспериментально-исследовательской части

По графикам сделать выводы об удачности эксперимента по внедрению кеширования в работу приложения.

Заключение [TODO]

Обязательно упомянуть, что у приложение будет развиваться в рамках микросервисной архитектуры. В частности, планируется добавить микросервисы для автоматической разметки текстов по описанному алгоритму и для работы с кешем.

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. Terryn 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 «Вильяме», 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. — Москва : Издательство Юрайт, 2021. — 178 с. — (Высшее образование). ISBN 978-5-534-13996-9

16. Ньюмен, С. Создание микросервисов, 2-е издание.: Пер. с англ. — СПб.: Издательство «Питер», 2023. — 624 с.: ил. ISBN 978-5-4461-1145-9

17. gRPC: A high performance, open source universal RPC framework [Электронный ресурс]. — Режим доступа: `https://grpc.io/` (дата обращения: 27.03.2023).

18. Swagger: API Documentation & Design Tools for Teams [Электронный ресурс]. — Режим доступа: `https://swagger.io/` (дата обращения: 15.04.2023).

19. Roy Thomas Fielding Architectural Styles and the Design of Network-based Software Architectures [Электронный ресурс]. — Режим доступа: `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm` (дата обращения: 27.03.2023).

20. Иванова Е.В., Цымблер М.Л. Обзор современных систем обработки временных рядов // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2020. Т. 9, № 4. С. 79–97. DOI: 10.14529/cmse200406.

21. 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

22. 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).

23. Seemann, M. and van Deursen, S. Dependency Injection. Principles, Practices, and Patterns. — Manning, 2019. — 552 с. ISBN 978-1-6172-9473-0

24. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/ (дата обращения: 15.04.2023).

25. Государственный реестр сертифицированных средств защиты информации [Электронный ресурс]. — Режим доступа: https://fstec.ru/tekhnicheskaya-zashchita-informatsii/dokumenty-po-sertifikatsii/153-sistema-sertifikatsii/591-gosudarstvennyj-reestr-sertifitsirovannykh-sredstv-zashchity-informatsii-n-ross-ru-0001-01bi00 (дата обращения: 15.04.2023).

26. Redis. The open source, in-memory data store used by millions of developers as a database, cache, streaming engine, and message broker. [Электронный ресурс]. — Режим доступа: https://redis.io/ (дата обращения: 15.04.2023).

27. InfluxData: InfluxDB Times Series Data Platform [Электронный ресурс]. — Режим доступа: https://www.influxdata.com/ (дата обращения: 15.04.2023).

28. O'Neil P., Cheng E., Gawlick D., O'Neil E. The log-structured merge-tree (LSM-tree) // Acta Informatica. 1996. Vol. 33. P. 351–385.

29. InfluxDB Query Language [Электронный ресурс]. — Режим доступа: https://docs.influxdata.com/influxdb/v1.3/query_language/ (дата обращения: 15.04.2023).

30. The Go Programming Language [Электронный ресурс]. — Режим доступа: https://go.dev/ (дата обращения: 15.04.2023).

31. Gin Web Framework [Электронный ресурс]. — Режим доступа: https://gin-gonic.com/ (дата обращения: 15.04.2023).

32. OpenAPI Specification - Version 3.0.3 - Swagger [Электронный ресурс]. — Режим доступа: https://swagger.io/specification/ (дата обращения: 15.04.2023).

33. Docker: Accelerated, Containerized Application Development [Электронный ресурс]. — Режим доступа: https://www.docker.com/ (дата обращения: 15.04.2023).

34. Docker Compose overview [Электронный ресурс]. — Режим доступа: https://docs.docker.com/compose/ (дата обращения: 15.04.2023).

35. PL/pgSQL — SQL Procedural Language [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/current/plpgsql.html (дата обращения: 16.04.2023).

Приложение А Алгоритм извлечения многокомпонентных терминов

Рисунок 8 – Функциональная схема работы системы извлечения многокомпонентных терминов, верхний уровень

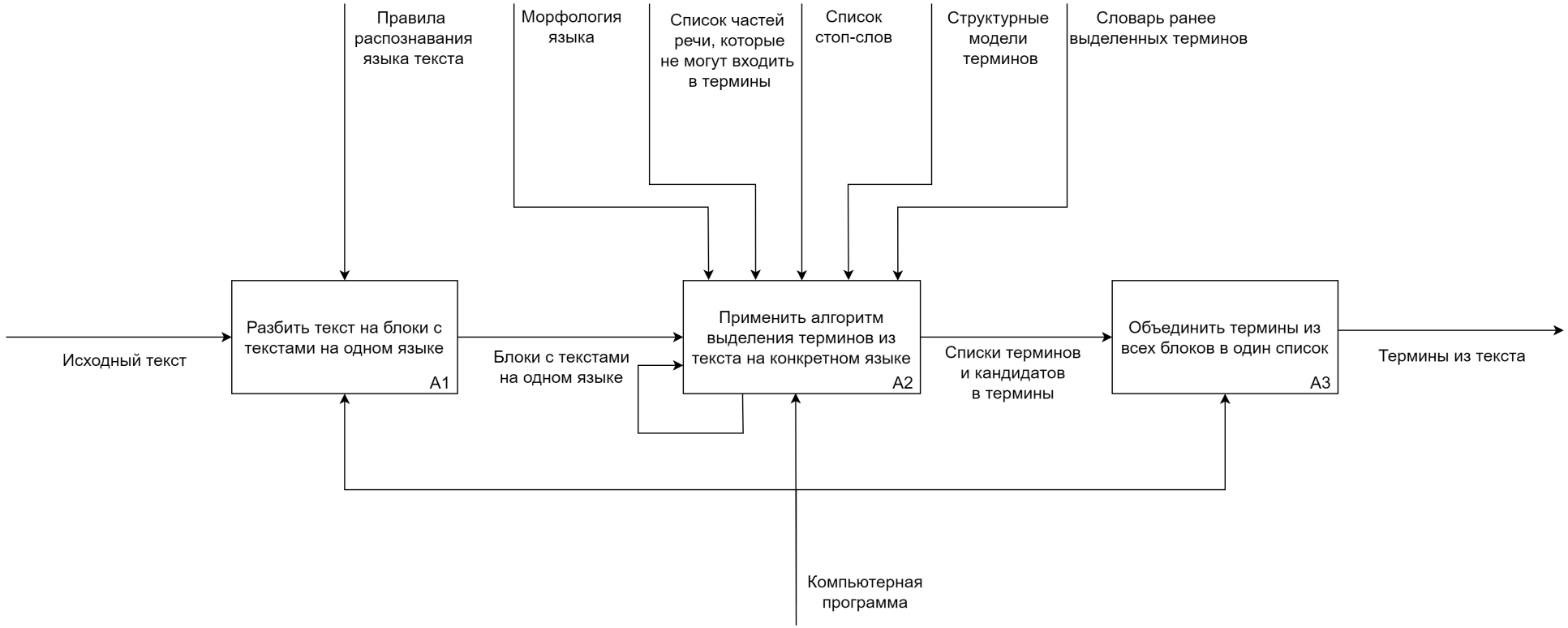

Рисунок 9 – Функциональная схема работы системы извлечения многокомпонентных терминов, декомпозиция уровня A0

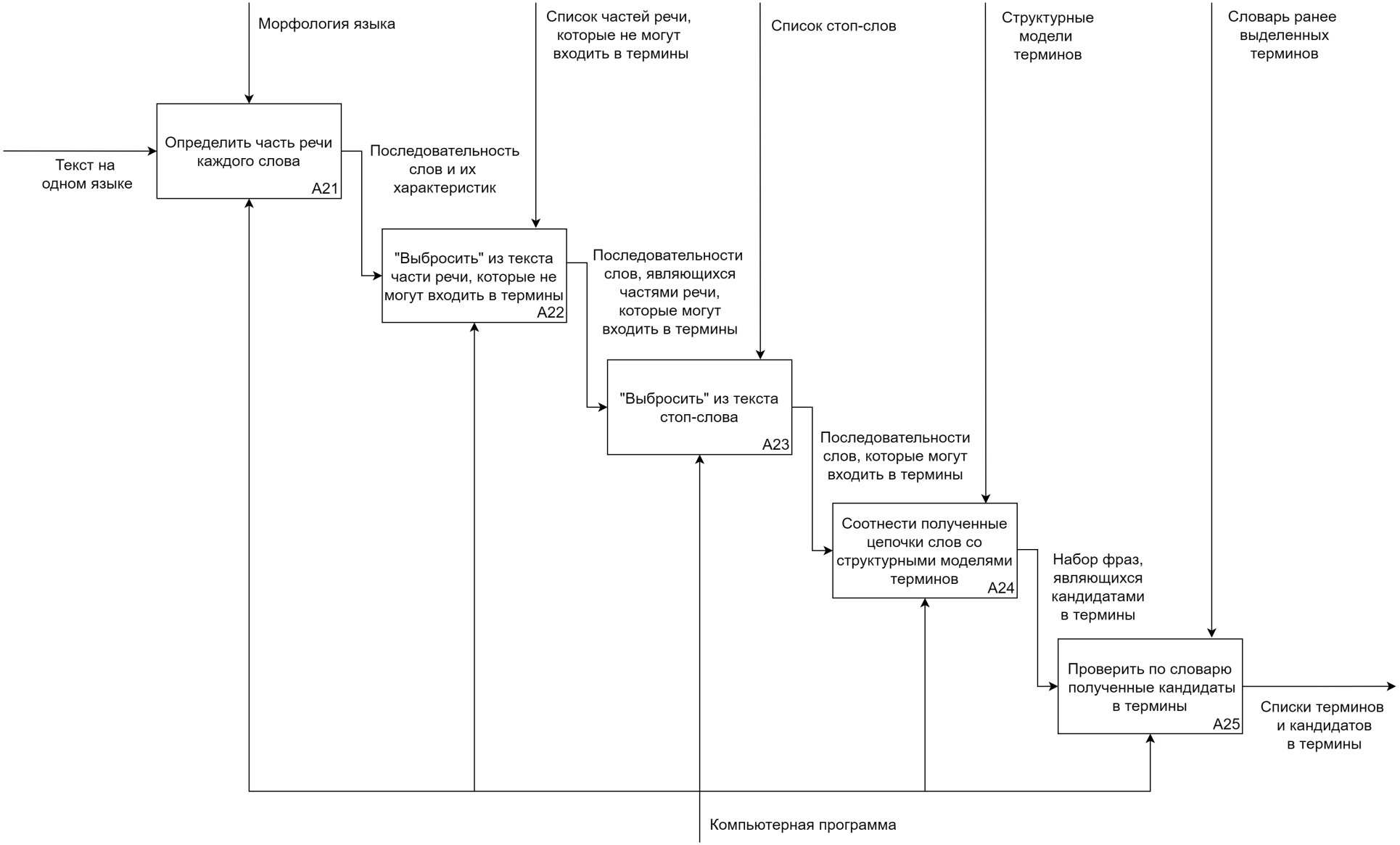

Рисунок 10 – Функциональная схема работы системы извлечения многокомпонентных терминов, декомпозиция уровня A2

Приложение Б Бизнес-сценарии

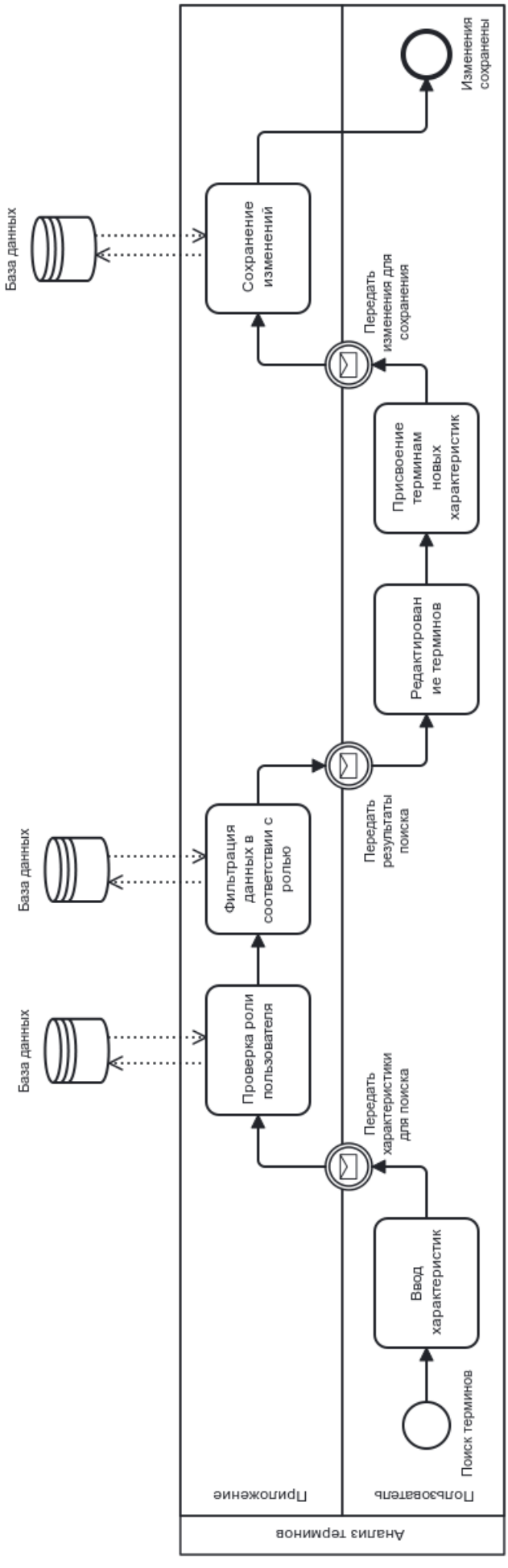

Рисунок 11 – Авторизация пользователя

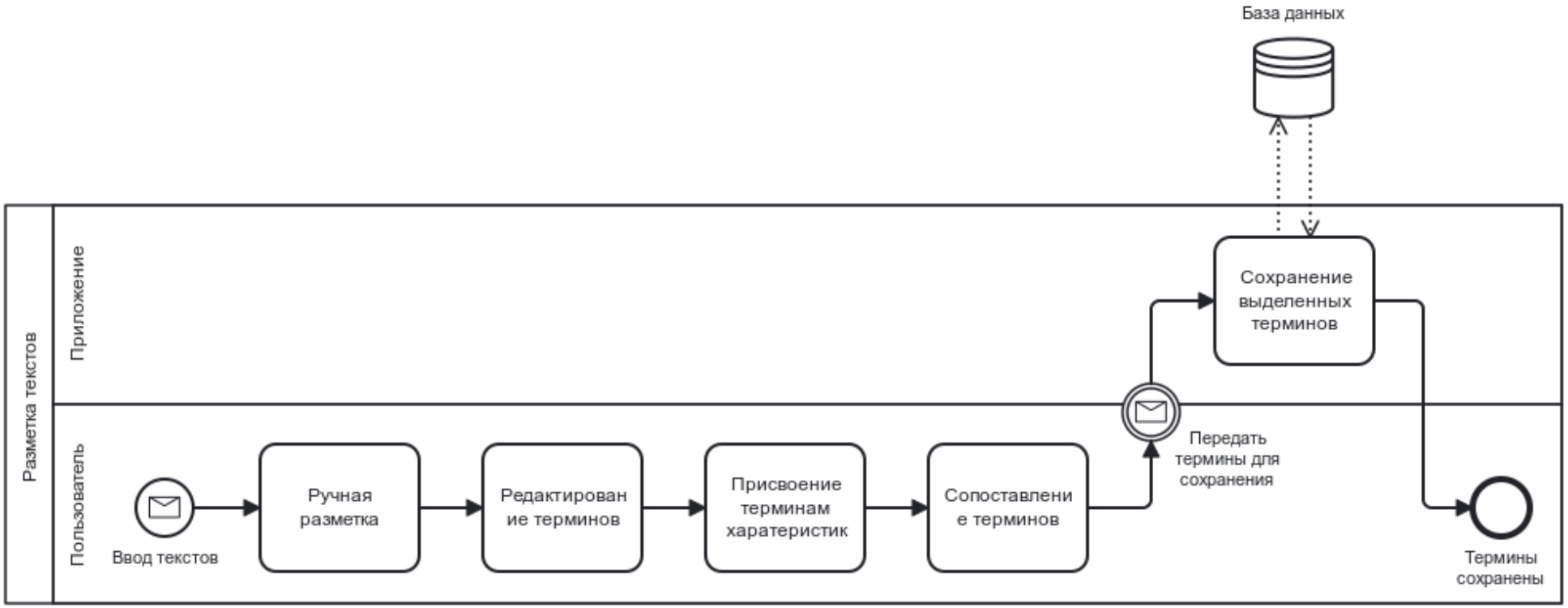

Рисунок 12 – Разметка текстов

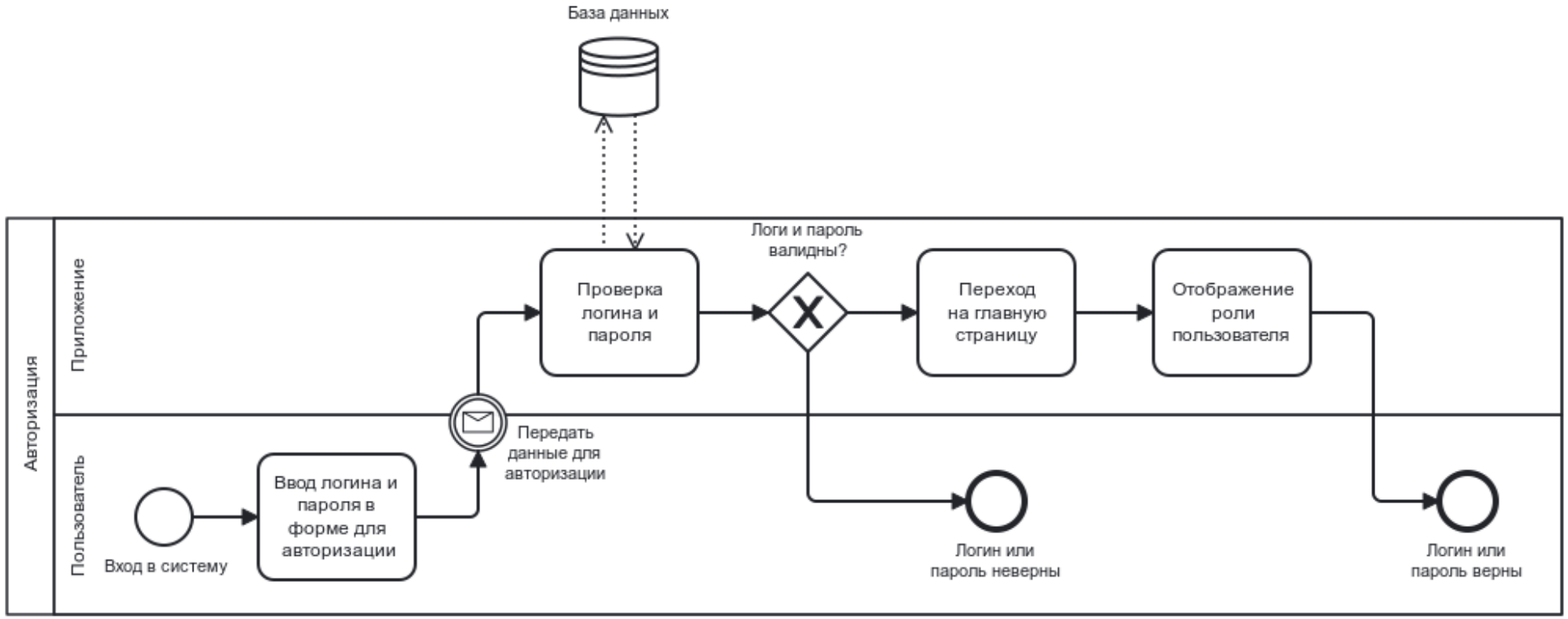

Рисунок 13 – Анализ терминов

Приложение В Инициализация ролевой модели базы данных

Листинг 1: Скрипт инициализации ролевой модели базы данных для постоянных таблиц

```
create role student;

grant select, insert, update on public.contexts to student;

grant select, insert, update, delete on public.properties to
   student;

grant select on information_schema.schemata to student;
grant select on information_schema.tables to student;

create role educator inherit;
grant student to educator;

grant delete on public.contexts to educator;

grant create on database :dbname to educator;
grant usage, create on schema public to educator;
grant references on all tables in schema public to educator;

create role admin with inherit CREATEDB CREATEROLE;
grant educator to admin;
alter database :dbname owner to admin;
grant usage, create on schema public to admin;
grant references, select, insert, update, delete on all tables in
   schema public to admin;
```

Приложение Г Хранимая процедура базы данных

Листинг 2: Скрипт создания хранимой процедуры для добавления нового слоя разметки текстов

```
CREATE OR REPLACE PROCEDURE public.create_layer_tables(layer text)
AS
$func$
DECLARE layer_name text;
BEGIN
    select (layer || '_layer') into layer_name;
    EXECUTE format(
            'create table if not exists %I.models(
                id int generated always as identity primary key,
                name text unique not null
            );

            create table if not exists %I.elements(
                id int generated always as identity primary key,
                name text unique not null
            );

            create table if not exists %I.units_ru(
                id int generated always as identity primary key,
                model_id int,
                foreign key (model_id) references %I.models(id),
                registration_date timestamp not null,
                text text unique not null
            );

            create table if not exists %I.units_en(
                id int generated always as identity primary key,
                model_id int,
                foreign key (model_id) references %I.models(id),
                registration_date timestamp not null,
```

```
31                  text text unique not null
32              );
33
34              create table if not exists %I.models_and_elems(
35                  model_id int,
36                  foreign key (model_id) references %I.models(id),
37                  elem_id int,
38                  foreign key (elem_id) references %I.elements(id),
39                  unique(model_id, elem_id)
40              );
41
42              create table if not exists %I.units_ru_and_en(
43                  unit_ru_id int,
44                  foreign key (unit_ru_id) references %I.units_ru(id)
    ,
45                  unit_en_id int,
46                  foreign key (unit_en_id) references %I.units_en(id)
    ,
47                  unique(unit_ru_id, unit_en_id)
48              );
49
50              create table if not exists %I.properties_and_units_ru(
51                  property_id int,
52                  foreign key (property_id) references public.
    properties(id),
53                  unit_id int,
54                  foreign key (unit_id) references %I.units_ru(id),
55                  unique(property_id, unit_id)
56              );
57
58              create table if not exists %I.properties_and_units_en(
59                  property_id int,
60                  foreign key (property_id) references public.
    properties(id),
61                  unit_id int,
```

```
62                  foreign key (unit_id) references %I.units_en(id),
63                  unique(property_id, unit_id)
64              );
65
66              create table if not exists %I.contexts_and_units_ru(
67                  context_id int,
68                  foreign key (context_id) references public.contexts
    (id),
69                  unit_id int,
70                  foreign key (unit_id) references %I.units_ru(id),
71                  unique(context_id, unit_id)
72              );
73
74              create table if not exists %I.contexts_and_units_en(
75                  context_id int,
76                  foreign key (context_id) references public.contexts
    (id),
77                  unit_id int,
78                  foreign key (unit_id) references %I.units_en(id),
79                  unique(context_id, unit_id)
80              );
81
82              create table if not exists %I.users_and_units_ru(
83                  user_id int,
84                  foreign key (user_id) references public.users(id),
85                  unit_id int,
86                  foreign key (unit_id) references %I.units_ru(id),
87                  unique(user_id, unit_id)
88              );
89
90              create table if not exists %I.users_and_units_en(
91                  user_id int,
92                  foreign key (user_id) references public.users(id),
93                  unit_id int,
94                  foreign key (unit_id) references %I.units_en(id),
```

```
95                  unique(user_id, unit_id)
96              );',
97              layer_name, layer_name, layer_name, layer_name,
98              layer_name, layer_name, layer_name, layer_name,
99              layer_name, layer_name, layer_name, layer_name,
100             layer_name, layer_name, layer_name, layer_name,
101             layer_name, layer_name, layer_name, layer_name,
102             layer_name, layer_name, layer_name, layer_name,
103             layer_name, layer_name, layer_name, layer_name,
104             layer_name, layer_name, layer_name, layer_name,
105             layer_name
106         );
107 END
108 $func$ LANGUAGE plpgsql;
109
110 CREATE OR REPLACE PROCEDURE public.
    grant_student_rights_to_layer_tables(layer text)
111 AS
112 $func$
113 DECLARE layer_name text;
114 BEGIN
115     select (layer || '_layer') into layer_name;
116     EXECUTE format(
117             'grant select, insert, update on %I.units_ru to student
                ;
118             grant select, insert, update on %I.units_en to student;
119             grant select, insert, update on %I.units_ru_and_en to
    student;
120             grant select, insert, update, delete on %I.
    properties_and_units_ru to student;
121             grant select, insert, update, delete on %I.
    properties_and_units_en to student;
122             grant select, insert, update on %I.
    contexts_and_units_ru to student;
```

```
123             grant select, insert, update on %I.
    contexts_and_units_en to student;
124             grant select, insert, update on %I.users_and_units_ru
    to student;
125             grant select, insert, update on %I.users_and_units_en
    to student;
126             grant select on %I.models to student;
127             grant select on %I.elements to student;
128             grant select on %I.models_and_elems to student;',
129             layer_name, layer_name, layer_name, layer_name,
130             layer_name, layer_name, layer_name, layer_name,
131             layer_name, layer_name, layer_name, layer_name
132         );
133 END
134 $func$ LANGUAGE plpgsql;
135
136 CREATE OR REPLACE PROCEDURE public.
    grant_educator_rights_to_layer_tables(layer text)
137 AS
138 $func$
139 DECLARE layer_name text;
140 BEGIN
141     select (layer || '_layer') into layer_name;
142     EXECUTE format(
143             'grant insert, update, delete on %I.models to educator;
144             grant insert, update, delete on %I.elements to educator
    ;
145             grant insert, update, delete on %I.models_and_elems to
    educator;',
146             layer_name, layer_name, layer_name
147         );
148 END
149 $func$ LANGUAGE plpgsql;
150
```

```
151 CREATE OR REPLACE PROCEDURE public.
       grant_admin_rights_to_layer_tables(layer text)
152 AS
153 $func$
154 DECLARE layer_name text;
155 BEGIN
156     select (layer || '_layer') into layer_name;
157     EXECUTE format(
158             'grant usage, create on schema %I to admin;
159             grant select, insert, update, delete on all tables in
       schema %I to admin;',
160             layer_name, layer_name
161         );
162 END
163 $func$ LANGUAGE plpgsql;
164
165 CREATE OR REPLACE PROCEDURE public.grant_rights_to_layer_tables(
       layer text)
166 AS
167 $func$
168 BEGIN
169     EXECUTE format(
170             E'call public.grant_student_rights_to_layer_tables(\'%s
                \');
171             call public.grant_educator_rights_to_layer_tables(\'%s\
       ');
172             call public.grant_admin_rights_to_layer_tables(\'%s\');
       ',
173             layer, layer, layer
174         );
175 END
176 $func$ LANGUAGE plpgsql;
177
178 CREATE OR REPLACE PROCEDURE public.create_layer(layer text)
179 AS
```

```
$func$
BEGIN
    EXECUTE format(
            E'set role educator;
            create schema if not exists %I;
            call public.create_layer_tables(\'%s\');
            call public.grant_rights_to_layer_tables(\'%s\');',
            (layer || '_layer'), layer, layer
        );
END
$func$ LANGUAGE plpgsql;
```

Приложение Д Сборка приложения

Листинг 3: Dockerfile для сервиса основного приложения

```
# syntax=docker/dockerfile:1

## Build
FROM golang:1.20.2-alpine3.17 AS build
WORKDIR /app

# Install dependencies
COPY go.mod ./
COPY go.sum ./
RUN go mod download

# Copy source code
COPY cmd/main.go ./
COPY ./internal ./internal
COPY ./pkg ./pkg

# Build the binary
RUN go build -o /backend

## Deploy
FROM scratch

# Copy our static executable
COPY --from=build /backend /backend
COPY backend.env /

EXPOSE ${BACKEND_PORT}
# USER nonroot:nonroot

# Run the binary
ENTRYPOINT ["/backend"]
```

Приложение Е Развёртывание приложения

Листинг 4: Конфигурация развертывания приложения

```
version: '3.8'

services:
    influxdb: # TODO: configure dashboards
        container_name: ${INFLUXDB_CONTAINER_NAME}
        image: influxdb:2.6.1-alpine
        environment:
            DOCKER_INFLUXDB_INIT_MODE: setup
            DOCKER_INFLUXDB_INIT_USERNAME: ${INFLUXDB_USERNAME}
            DOCKER_INFLUXDB_INIT_PASSWORD: ${INFLUXDB_PASSWORD}
            DOCKER_INFLUXDB_INIT_ORG: ${INFLUXDB_ORG}
            DOCKER_INFLUXDB_INIT_BUCKET: default
            DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: ${INFLUXDB_TOKEN}
        restart: always
#         healthcheck:
#             # test: "curl -f http://localhost:8086/ping"
#             test: ${INFLUXDB_SERVICE_HEALTHCHECK_CMD}
#             interval: 3s
#             timeout: 10s
#             retries: 5
        ports:
            - ${INFLUXDB_PORT}:${INFLUXDB_PORT}
    postgres:
        container_name: ${POSTGRES_CONTAINER_NAME}
        image: postgres:15.2-alpine3.17
        environment:
            POSTGRES_USER: ${POSTGRES_USER}
            POSTGRES_DB: ${POSTGRES_DBNAME}
            POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
            PGDATA: /data/postgres
        restart: always
```

```
        healthcheck:
            test: [ "CMD-SHELL", "pg_isready -U postgres" ]
            interval: 5s
            timeout: 5s
            retries: 5
        volumes:
            - postgres:/var/lib/postgresql/data
        ports:
            - ${POSTGRES_PORT}:${POSTGRES_PORT}
        networks:
            - persistent_bridge_network
        depends_on:
            - influxdb
    pgadmin:
        container_name: pgadmin4
        image: dpage/pgadmin4
        restart: always
        environment:
            PGADMIN_DEFAULT_EMAIL: admin@admin.com
            PGADMIN_DEFAULT_PASSWORD: root
        ports:
            - "5050:80"
        depends_on:
            postgres:
                condition: service_healthy
    redis:
        container_name: redis
        image: redis:7.0.8-alpine3.17
        environment:
            REDIS_HOST: ${REDIS_HOST}
            REDIS_PORT: ${REDIS_PORT}
            REDIS_PASSWORD: ${REDIS_PASSWORD}
        restart: always
        healthcheck:
            test: [ "CMD", "redis-cli", "ping" ]
```

```
            interval: 5s
            timeout: 5s
            retries: 5
        ports:
            - ${REDIS_PORT}:${REDIS_PORT}
        command: redis-server --save 20 1 --loglevel warning --
           requirepass ${REDIS_PASSWORD}
        volumes:
            - redis:/data
        depends_on:
            postgres:
                condition: service_healthy
    backend:
        container_name: backend
        build:
            context: ./
            dockerfile: api.Dockerfile
        image: golang-backend
        restart: always
        ports:
            - ${BACKEND_PORT}:${BACKEND_PORT}
        depends_on:
            influxdb:
                condition: service_started
            redis:
                condition: service_healthy
            postgres:
                condition: service_healthy
volumes:
    redis:
        driver: local
    postgres:
        driver: local
networks:
    persistent_bridge_network:
```

```
101         driver: bridge
```

Приложение Ж Презентация