



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
КАФЕДРА

ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ (ИУ)
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

КУРСОВАЯ РАБОТА

НА ТЕМУ:

*Разработка базы данных для АРМ разметчика
параллельного корпуса технических текстов*

Студент

ИУ7-64Б
(группа)

(подпись, дата)

Рунов К.А.

(И.О. Фамилия)

Руководитель курсового проекта

(подпись, дата)

Строганов Ю.В.

(И.О. Фамилия)

2024 г.

РЕФЕРАТ

Отчет 67 с., 11 рис., 18 табл., 29 источн., 2 прил.

КОРПУСНАЯ ЛИНГВИСТИКА, ПАРАЛЛЕЛЬНЫЕ КОРПУСА ТЕКСТОВ, ТЕРМИНОЛОГИЧЕСКАЯ РАЗМЕТКА, БАЗА ДАННЫХ, РЕЛЯЦИОННАЯ МОДЕЛЬ, POSTGRESQL, KEYDB, LOCUST

Цель работы — разработка базы данных для автоматизации рабочего места разметчиков параллельного корпуса технических текстов.

В данной работе был проведен анализ предметной области корпусов текстов, рассмотрены существующие модели данных и была выбрана модель, на основе которой проводилась дальнейшая разработка базы данных. Были спроектированы и реализованы 10 сущностей базы данных, ограничения целостности, ролевая модель на уровне базы данных. Было описан интерфейс для доступа к базе данных и было разработано Web-приложение для взаимодействия с базой данных. Было проведено исследование зависимости времени ответа от количества запросов в секунду и проведен сравнительный анализ эффективности реализаций приложения с использованием дополнительного кеширования и без него. В результате исследования был получен следующий результат — при 752 запросах в секунду использование кеша позволяет ускорить среднее время ответа более, чем в 3 раза.

СОДЕРЖАНИЕ

РЕФЕРАТ	3
ОПРЕДЕЛЕНИЯ	6
ВВЕДЕНИЕ	8
1 Аналитический раздел	9
1.1 Анализ предметной области	9
1.2 Существующие решения	13
1.3 Формализация задачи	14
1.4 Формализация данных	14
1.5 Формализация и описание пользователей	16
1.6 Модели данных	17
1.6.1 Реляционная	17
1.6.2 Инвертированные списки	18
1.6.3 Иерархическая	19
1.6.4 Сетевая	20
1.6.5 Постреляционная	21
1.7 Выбор модели данных	21
2 Конструкторский раздел	23
2.1 Проектирование базы данных	23
2.2 Описание сущностей	23
2.3 Описание ограничений целостности	28
2.4 Описание функций, процедур и триггеров	32
2.5 Описание ролевой модели	33
3 Технологический раздел	35
3.1 Выбор средств реализации	35
3.2 Описание реализаций	36
3.2.1 Сущности базы данных	36
3.2.2 Ограничения целостности базы данных	40
3.2.3 Ролевая модель на уровне базы данных	45

3.2.4	Функции, процедуры и триггеры	46
3.3	Тестирование функций	46
3.4	Интерфейс доступа к базе данных	48
4	Исследовательский раздел	49
4.1	Описание исследования	49
4.2	Технические характеристики	53
4.3	Результаты исследования	53
	ЗАКЛЮЧЕНИЕ	60
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	63
	ПРИЛОЖЕНИЕ А	64
	ПРИЛОЖЕНИЕ Б	67

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

База данных — это некоторый набор перманентных (постоянно хранимых) данных, используемых прикладными программными системами какого-либо предприятия. [1, с. 51]

Домен — это подмножество значений некоторого типа данных, имеющих определенный смысл. [2]

Атрибут отношения — это пара вида $\langle \text{имя_атрибута}, \text{имя_домена} \rangle$. Имена атрибутов должны быть уникальны в пределах отношения. [2]

Схема отношения — это именованное множество упорядоченных пар $\langle \text{имя_атрибута}, \text{имя_домена} \rangle$. Степенью или «арностью» схемы отношения является мощность этого множества. [2]

Кортеж, соответствующий данной схеме отношения — это множество упорядоченных пар $\langle \text{имя_атрибута}, \text{значение_атрибута} \rangle$, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. Значение атрибута должно быть допустимым значением домена, на котором определен данный атрибут. Степень кортежа совпадает со степенью соответствующей схемы отношения. [2]

Отношение, определенное на множестве из N доменов содержит две части: заголовок (схему отношения) и тело (множество из M кортежей). Значения N и M называются соответственно степенью и кардинальностью отношения. [2]

Потенциальный ключ — это непустое подмножество множества атрибутов схемы отношения, обладающее свойствами уникальности (в отношении нет двух различных кортежей с одинаковыми значениями потенциального ключа) и избыточности (никакое из собственных подмножеств множества потенциального ключа не обладает свойством уникальности). [2]

Первичный ключ — выбранный потенциальный ключ. [2]

Альтернативный ключ — потенциальный ключ, не являющийся первичным. [2]

Внешний ключ в отношении R_2 — это непустое подмножество множества атрибутов FK этого отношения, такое, что:

- а) существует отношение R1 с потенциальным ключом СК;
- б) каждое значение внешнего ключа FK в текущем значении отношения R2 обязательно совпадает со значением ключа СК некоторого кортежа в текущем значении отношения R1. [2]

Реляционная база данных — это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных. [2]

Система управления базой данных (СУБД) представляет собой программное обеспечение, которое управляет всем доступом к базе данных. [1, с. 87]

ВВЕДЕНИЕ

В современном мире немаловажное значение имеет корпусная лингвистика. Корпуса текстов находят применение в различных областях — в машинном переводе, в разработке словарей, в лингвистических исследованиях. Для того, чтобы из корпуса текстов можно было извлекать пользу, тексты в нем должны быть размечены. Существуют алгоритмы, позволяющие автоматически производить разметку, но для проверки ее корректности все равно требуется вмешательство человека.

На данный момент не существует открытых параллельных корпусов технических текстов. Также нет открытых информационных систем, позволяющих одновременно

- производить разметку текста в параллельном корпусе,
- производить поиск по параллельному корпусу,
- организовать удобную работу множества разметчиков.

Создание такой информационной системы позволит во многом автоматизировать рабочее место разметчиков параллельного корпуса.

Целью данной работы является разработка базы данных для автоматизации рабочего места разметчиков параллельного корпуса технических текстов.

Задачи курсового проекта:

- провести анализ предметной области параллельных корпусов текстов;
- спроектировать сущности базы данных и ограничения целостности АРМ разметчика корпуса технических текстов;
- выбрать средства реализации базы данных и приложения;
- разработать сущности базы данных и реализовать ограничения целостности базы данных;
- описать интерфейс доступа к базе данных;
- исследовать зависимость времени ответа от количества запросов в секунду.

1 Аналитический раздел

В данном разделе будет проведен анализ предметной области корпусов текстов, будет формализована задача и данные, описаны пользователи, которые будут взаимодействовать с проектируемым приложением к базе данных, рассмотрены существующие модели данных и будет выбрана одна из моделей для дальнейшей разработки базы данных на ее основе.

1.1 Анализ предметной области

Корпусная лингвистика — это раздел компьютерной лингвистики, который занимается разработкой принципов построения и использования корпусов текстов с помощью компьютерных технологий. Лингвистические корпуса представляют собой структурированные массивы данных, которые используются для изучения языковых единиц в текстах. В рамках корпуса существует поисковая система, которая позволяет находить необходимые языковые единицы и примеры их употребления благодаря технологии текстовой разметки. В свою очередь разметка может быть ручной и автоматической. [3]

Виды корпусов текстов

В таблице 1 приведена классификация корпусов текстов по разным признакам.

Признак	Типы корпусов
Цель	Многоцелевые, специализированные
Параллельность	Параллельные, сопоставимые
Динамичность	Динамические (мониторные), статические
Разметка	Размеченные, неразмеченные
Характер разметки	Морфологические, синтаксические, семантические, анафорические, просодические и т. д.
Объем текстов	Полнотекстовые, «фрагментнотекстовые»

Таблица 1 – Классификация корпусов [4, с. 57]

Корпус технических текстов, для которого будет разрабатываться база

данных в настоящей работе, является специализированным, параллельным, многоязыковым, динамическим (будет постоянно пополняться).

Параллельные корпуса

Параллельные корпуса — корпуса, представляющие собой множество текстов-оригиналов, написанных на каком-либо исходном языке, и текстов — переводов этих исходных текстов на один или несколько других языков [4, с. 61].

При подготовке параллельных корпусов и разработке программ для их обработки, требуется выровнять тексты — установить соответствие между фрагментами текста оригинала и текста перевода. Для решения этой задачи существуют различные методы автоматического выравнивания текстов по предложениям, грамматическим конструкциям, терминам, словам и словосочетаниям. [4, с. 61]

Ниже приведен пример выравнивания текстов на уровне предложений.

1	THE PLAY — for which Briony had designed the posters, programs and tickets, constructed the sales booth out of a folding screen tipped on its side, and lined the collection box in red crepe paper — was written by her in a two-day tempest of composition, causing her to miss a breakfast and a lunch.	Пьеса, для которой Брайони рисовала афиши, делала программки и билеты, сооружала из ширмы кассовую будку и обклеивала коробку для денежных сборов гофрированной красной бумагой, была написана ею за два дня в порыве вдохновения, заставлявшего ее забывать даже о еде.
2	When the preparations were complete, she had nothing to do but contemplate her finished draft and wait for the appearance of her cousins from the distant north.	Когда приготовления закончились, ей не оставалось ничего, кроме как созерцать свое творение и ждать появления кузенов и кузины, которые должны были прибыть с далекого севера.

Таблица 2 – Пример выравнивания текстов на уровне предложений [4, с. 62]

Проблемы определения границ терминов

При попытке проведения автоматического выравнивания на уровне терминов, возникает ряд проблем. Среди них выделяют [3]:

- неправильное определение границ терминов-словосочетаний, состоящих из двух и более слов и составных терминов;
- распознавание составных терминов и терминов-словосочетаний, состоящих из двух и более слов; в частности, распознавание лексической единицы как части составного термина или как свободной лексической единицы;
- определение лексической единицы как термина в зависимости от контекста и тематики текста, в котором данная лексическая единица употребляется;
- объемные списки терминов-кандидатов, которые необходимо проверять вручную, поскольку частота не является достаточным критерием для оценки того, является ли выделенное слово термином или нет.

Точность определения границ термина при автоматической разметки является одной из основных лингвистических задач, а отсутствие на сегодняшний день информационных систем для автоматической разметки русскоязычных текстов делает актуальным разработку таковой.

Типы текстовых разметок

Существует множество типов текстовых разметок. Большинство современных корпусов относятся к корпусам морфологического или синтаксического вида [4, с. 56].

Далее подробнее будут рассмотрены структурная и семантическая разметки, поскольку они являются основными типами разметки в параллельном корпусе технических текстов.

Структурная разметка

Структурная разметка предназначена для выделения структурных элементов текста (том, книга, часть, глава, действие, сноска, ремарка, стих, а также: абзац, предложение, словоформа и текстоформа — таблица, формула и др.) [5]. В контексте учебно-научных текстов структурная разметка используется для выделения названия статьи, авторов, оглавления, предисловия, введения и т.д.

На рисунке 1 представлена структурная схема элементов учебно-научного текста.

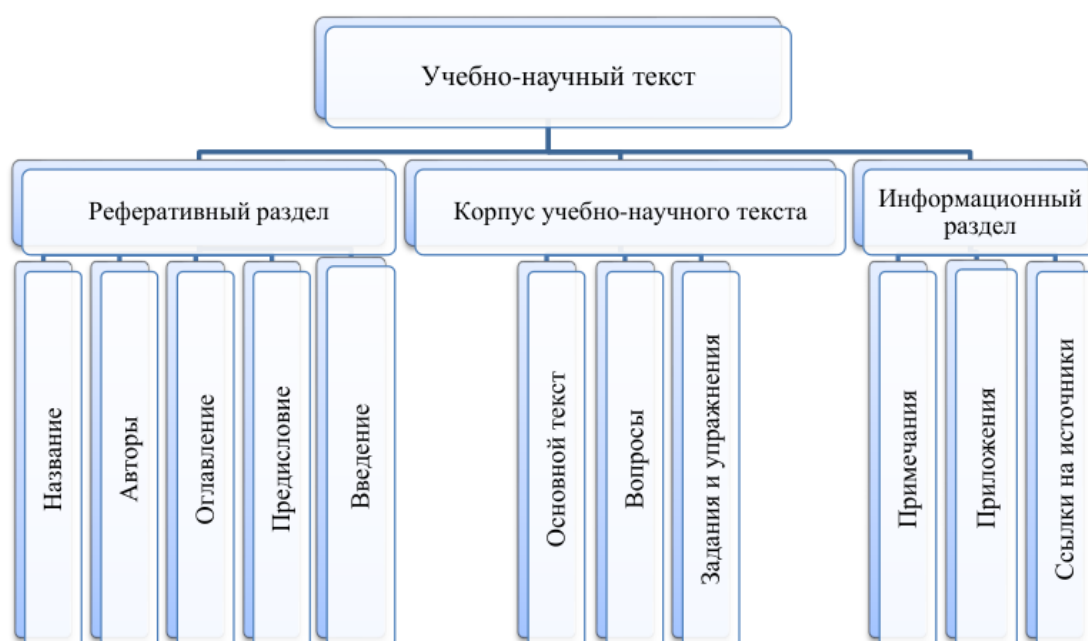


Рисунок 1 – Структурные элементы учебно-научных текстов [6]

Семантическая разметка

Семантическая разметка помогает установить контекст высказывания и устранить двусмысленность. Основная цель семантической разметки — «формализовать» значения слов и сделать тексты пригодными для машинной обработки. [7]

Существует множество видов семантических разметок. Один из вариантов русской семантической разметки представлен на сайте Центра компьютерных и корпусных языковых исследований Ланкастерского университета [8, 4, с. 51]:

A	Общие понятия
A1	Общие понятия
A1.1.1	Обычные действия / Изголовление ч.-л.
A1.1.1	Повреждение и разрушение
A1.2	Пригодность
A1.3	Осторожность
...
B	Тело человека
B1	Анатомия и физиология
B2	Здоровье и болезнь
...
C	Искусства и ремесла
...

Таблица 3 – Русская семантическая разметка [8]

В научно-технических текстах для семантической разметки могут использоваться семантические падежи Ч. Филлмора [7].

1.2 Существующие решения

В современном мире существует множество параллельных корпусов (Opus, Linguae, MyMemory, Glosbe, Reverso, TAUS Data Cloud и др.) [9], сервисов для автоматического выравнивания текстов (Hunalign, Euclid, Abbyy Aligner, Trados, Winalign, Wordfast tools, Giza++ и др.) [4, с. 62], служб, позволяющих создавать собственные корпусы и производить в них поиск (SketchEngine [10], NoSketchEngine [11]); инструментов для автоматического извлечения терминов (TerMine, TermExtraction, Terminology Extraction) [3] и прочих инструментов для работы с корпусами текстов (OpenCorpora [12]).

Но на данный момент не существует открытых параллельных корпусов технических текстов [13]. Также нет открытых информационных систем, позволяющих одновременно производить разметку текста в параллельном корпусе, производить поиск по параллельному корпусу и организовать удобную работу множества разметчиков.

1.3 Формализация задачи

В ходе выполнения курсовой работы необходимо спроектировать и разработать базу данных для хранения документов — технических текстов на разных языках, их разметок, и информации о пользователях базы данных.

Для взаимодействия с базой данных необходимо разработать интерфейс, предоставляющий возможности

- добавления новых документов в базу данных,
- добавления новых разметок в базу данных,
- произведения поиска по текстам и разметкам, хранящимся в базе данных.

1.4 Формализация данных

Разрабатываемая база данных должна хранить информацию о следующих сущностях:

- пользователь;
- документ;
- метаданные о документе — автор;
- задание на разметку;
- структурная разметка;
- терминологическая разметка.

На рисунке 2 представлена диаграмма сущностей разрабатываемой базы данных в нотации Чена.

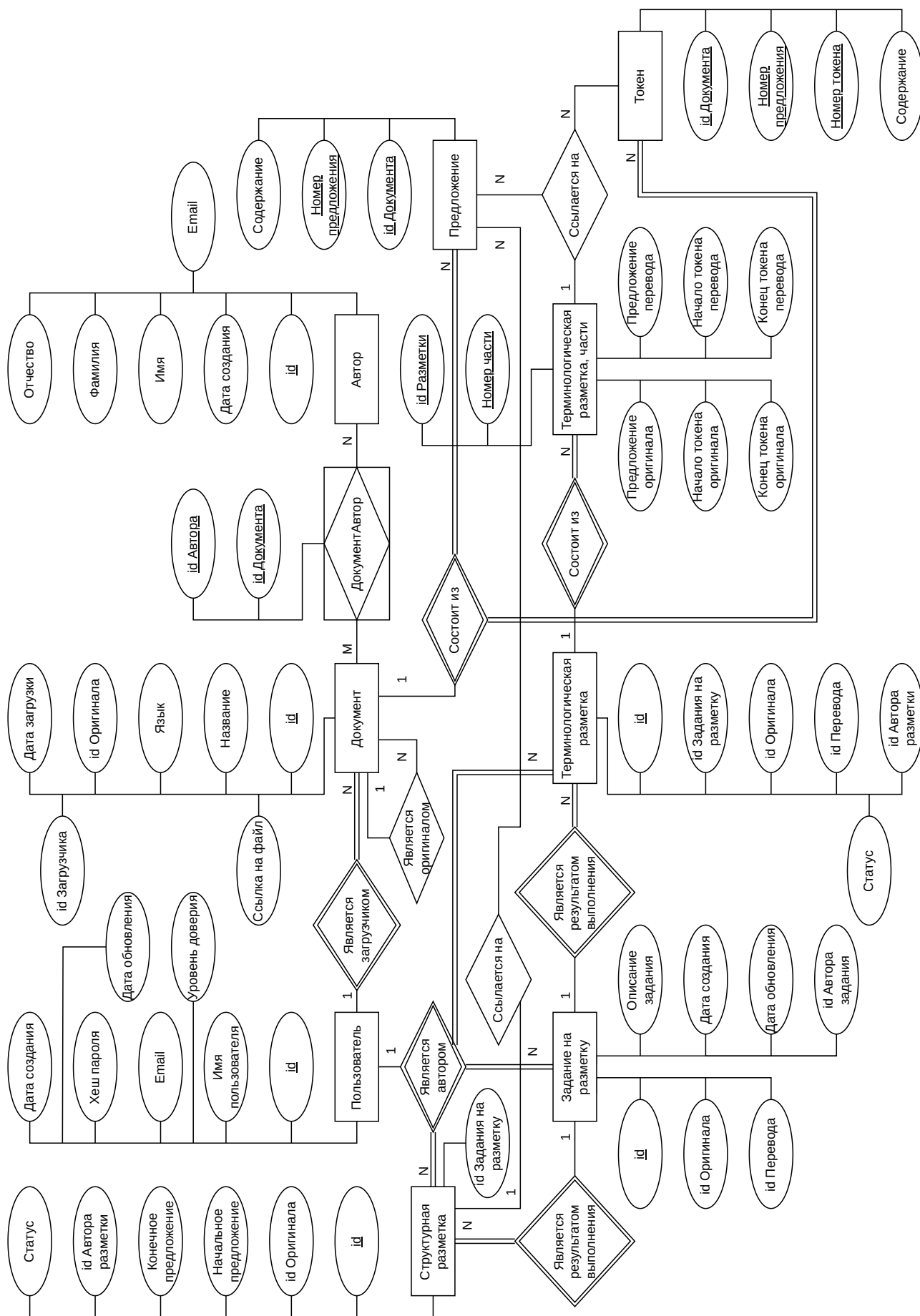


Рисунок 2 – ER-диаграмма в нотации Чена

1.5 Формализация и описание пользователей

Взаимодействовать с проектируемым приложением к базе данных будут три вида пользователей:

- администраторы;
- модераторы;
- пользователи.

Администратор имеет полный доступ к данным: может добавлять, удалять тексты; добавлять, удалять, изменять разметки.

Модераторы производят проверку разметки.

Пользователи имеют возможность осуществлять разметку, которая после своего создания должна пройти модерацию перед публикацией. Также пользователи могут искать нужные разметку и текст среди уже существующих.

На рисунке 3 приведена диаграмма вариантов использования проектируемого приложения к базе данных.

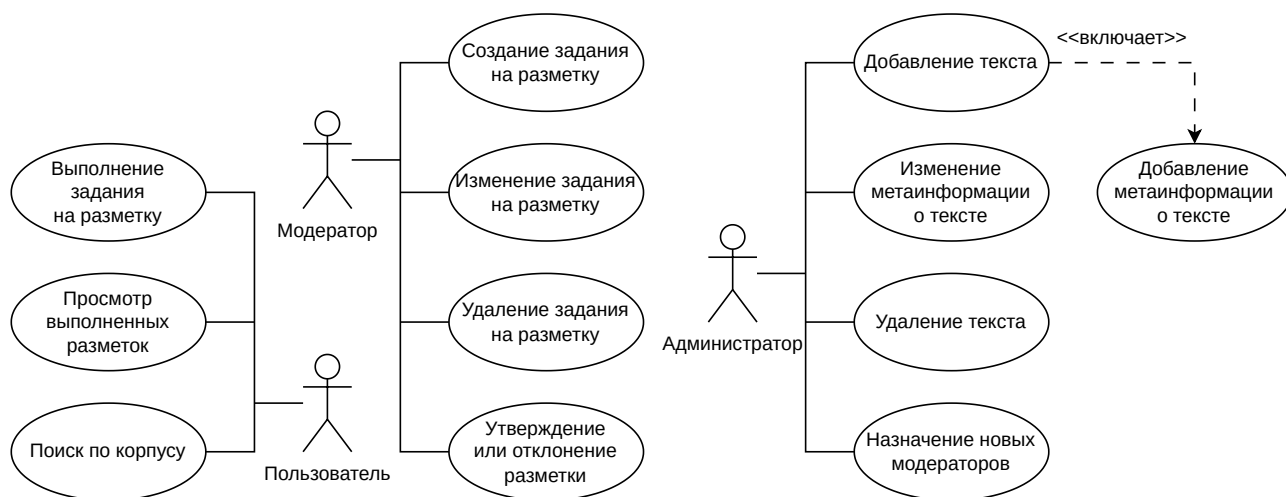


Рисунок 3 – Диаграмма вариантов использования

1.6 Модели данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Упомянутые объекты позволяют моделировать структуру данных, а операторы — поведение данных. [1]

Дейт выделяет три составляющих моделей данных: структурную, манипуляционную и целостную. [14]

1.6.1 Реляционная

Структурная часть реляционной модели описывает, из каких объектов состоит реляционная модель. Постулируется, что основной структурой данных, используемой в реляционной модели, являются нормализованные «парные» отношения. [1, 2]

В реляционной модели каждый кортеж любого отношения должен отличаться от другого кортежа этого отношения. [1, 2]

Манипуляционная часть реляционной модели описывает два эквивалентных способа манипулирования реляционными данными — реляционную алгебру и реляционное исчисление. [2]

В целостной части реляционной модели фиксируются два базовых требования целостности, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность ссылок. [2]

Поддержание целостности сущностей обеспечивается средствами СУБД и осуществляется с помощью двух ограничений:

- 1) при добавлении записей в таблицу проверяется уникальность их первичных ключей,
- 2) не допускается изменение значений атрибутов, входящих в первичный ключ. [2]

Требование целостности ссылок состоит в следующем: Для каждого значения внешнего ключа, появляющегося в дочернем отношении, в роди-

тельском отношении должен найтись кортеж с таким же значением первичного ключа. [2]

1.6.2 Инвертированные списки

Модель инвертированных списков похожа на реляционную, но на более низком уровне абстракции — хранимые таблицы, а также некоторые пути доступа к этим хранимым таблицам (в частности, некоторые индексы) напрямую доступны пользователю. Как и реляционная база данных, база данных инвертированных списков содержит множество «таблиц» или файлов, и эти «таблицы» или файлы разделены на строки (записи) и колонки (поля), как и в реляционном случае. [14, с. 388]

Однако, существуют и некоторые значительные различия:

- 1) Строки таблицы инвертированного списка упорядочены в физической последовательности, независимой от индексов. Поля внутри строк также упорядочены слева направо.
- 2) Определяется общий порядок для базы данных, где строки одной таблицы могут предшествовать строкам другой или чередоваться в определенном порядке.
- 3) Можно задать любое количество ключей поиска для таблицы, что позволяет как прямой, так и последовательный доступ на основе этих ключей. Индексы не являются прозрачными для пользователя, но их поддержка осуществляется СУБД.

В общем случае операторы манипуляции данными в модели инвертированных списков делятся на две широкие категории:

- 1) Операторы, которые устанавливают адресуемость к какой-либо записи в базе данных (операторы поиска, нахождения или определения местоположения).
- 2) Операторы, которые работают с записью по ранее установленному адресу.

Операторы поиска, в свою очередь, делятся на две подкатегории:

- а) Операторы, которые находят запись «из ниоткуда» — то есть операторы прямого поиска.
- б) Операторы, которые находят запись в терминах ее позиции относительно какого-либо ранее установленного адреса — то есть операторы относительного поиска. [14, с. 389]

Модель инвертированных списков не поддерживает общие правила целостности. Большинство ограничений должны обеспечиваться пользователем, включая ссылочную целостность. [14, с. 391]

1.6.3 Иерархическая

Иерархическая база данных состоит из упорядоченной коллекции экземпляров одного типа дерева.

Тип дерева включает один корневой тип записи и упорядоченную коллекцию зависимых (нижнего уровня) типов поддеревьев. Тип поддерева, в свою очередь, также состоит из одного типа записи — корневого типа поддерева — и упорядоченной коллекции зависимых типов поддеревьев нижнего уровня. Таким образом, весь тип дерева представляет собой иерархическую структуру типов записей. [14, с. 406]

Основное различие между иерархической и реляционной структурой заключается в следующем: в иерархической базе данных информация, которая в реляционной базе данных представлена внешними ключами, представлена связями «родитель-ребенок». [14, с. 408]

Язык манипулирования данными в иерархической базе данных включает операторы для работы с данными в виде деревьев:

- Нахождение конкретного дерева в базе данных.
- Переход от одного дерева к следующему в иерархической последовательности.
- Перемещение от записи к записи внутри дерева по иерархическим путям.
- Вставка новой записи в дерево.

- Удаление указанной записи.

Эти операторы, как правило, работают на уровне записей (манипулируют отдельными записями (или строками) в базе данных, а не группами записей или целыми наборами данных). [14, с. 410]

Иерархическая модель автоматически поддерживает ссылочную целостность по правилу: ни один потомок не может существовать без своего родителя. Если удаляется родитель, система удаляет все поддерево. Потомок не может быть вставлен без существующего родителя. Это эквивалентно соблюдению правил внешнего ключа в реляционной модели. [14, с. 411]

1.6.4 Сетевая

Сетевая база данных состоит из набора записей и связей — точнее, из любого числа экземпляров нескольких типов записей и связей. Каждый тип связи включает два типа записей: родительский и дочерний. Каждый экземпляр связи состоит из одного экземпляра родительского типа записи и упорядоченного набора экземпляров дочернего типа записи. Для конкретного типа связи L с родительским типом записи P и дочерним типом записи C справедливо следующее:

- Каждый экземпляр P является родителем в одном экземпляре L .
- Каждый экземпляр C является дочерней записью в не более чем одном экземпляре L .

Таким образом, сетевую структуру данных можно рассматривать как расширенную форму иерархической структуры данных. Основное различие между ними заключается в следующем: в иерархической структуре у дочерней записи есть ровно один родитель; в сетевой структуре у дочерней записи может быть любое количество родителей. [14, с. 447]

Язык манипулирования данными в сетевой базе данных состоит из набора операторов для работы с данными, представленными в виде записей и связей. Примеры таких операторов включают следующее:

- Оператор для нахождения конкретной записи по значению какого-либо поля в этой записи;

- Оператор для перехода от родителя к его первому ребенку в какой-либо связи;
- Оператор для перехода от одного ребенка к следующему в какой-либо связи;
- Оператор для перехода от ребенка к родителю в какой-либо связи;
- Оператор для создания новой записи;
- Оператор для удаления существующей записи;
- И так далее.

Эти операторы обычно так же, как и в инвертированных списках, и в иерархической модели, работают на уровне записей.

Сетевая модель, как и иерархическая, включает встроенную поддержку ссылочной целостности через связи. Например, можно задать правило, что дочерняя запись не может быть вставлена без существующего родителя. [14, с. 452]

1.6.5 Постреляционная

Постреляционная модель расширяет реляционную модель. Она устраняет ограничение на атомарность данных, позволяя использовать поля с многозначными значениями. Значения этих полей могут содержать подзначения, и такой набор значений рассматривается как отдельная таблица, встроенная в основную таблицу. [2]

1.7 Выбор модели данных

Как видно по рисунку 2, соблюдения строгой иерархии на диаграмме сущностей разрабатываемой базы данных не наблюдается. В случае сетевой модели возникают проблемы с выбором «главного» элемента, с которого будет осуществляться обход, и с обработкой нескольких путей, ведущих от главного элемента к искомому в случае наличия связи «многие-ко-многим». Описанная ранее формализация данных лучше всего укладывается в рамки реляционной модели, поэтому в основе разрабатываемой базы данных будет

использоваться именно она. Помимо прочего, у реляционной модели есть еще ряд преимуществ — современные ее реализации могут автоматически обеспечивать целостность данных и оптимизировать запросы, а также поддерживают транзакционность.

Вывод

В данном разделе был проведен анализ предметной области корпусов текстов, была формализована задача и данные, описаны пользователи, которые будут взаимодействовать с проектируемым приложением к базе данных, рассмотрены существующие модели данных, и была выбрана реляционная модель.

2 Конструкторский раздел

В данном разделе будет представлена схема проектируемой базы данных, будут описаны сущности базы данных, ограничения целостности, ролевая модель и используемые функции, процедуры и триггеры.

2.1 Проектирование базы данных

На рисунке 4 представлена схема проектируемой базы данных.

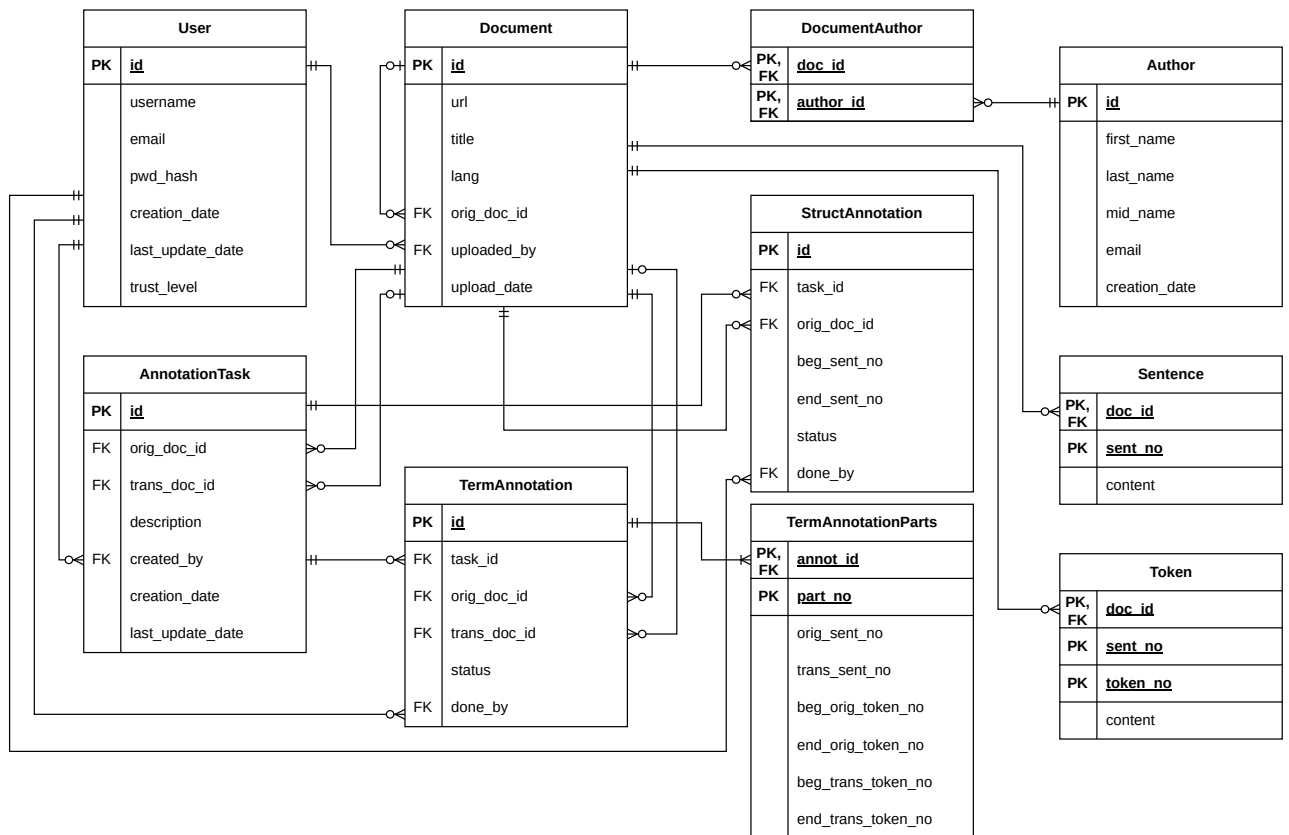


Рисунок 4 – ER-диаграмма проектируемой базы данных в нотации Мартина

2.2 Описание сущностей

В данном подразделе будут описаны десять сущностей проектируемой базы данных: User, Document, DocumentAuthor, Author, AnnotationTask, StructAnnotation, TermAnnotation, TermAnnotationPart, Sentence, Token.

User

Сущность User формализует пользователя базы данных и имеет следующие поля:

- id — идентификатор пользователя;
- username — имя пользователя;
- email — электронная почта пользователя;
- pwd_hash — хеш пароля пользователя;
- trust_level — уровень доверия пользователя (зависит от качества и количества совершенных разметок);
- creation_date — дата регистрации пользователя;
- last_update_date — дата последнего обновления информации о пользователе.

Document

Сущность Document формализует документ, хранимый в корпусе, и имеет следующие поля:

- id — идентификатор документа;
- url — ссылка на документ (в сети или файловой системе);
- title — название документа / статьи;
- lang — язык документа;
- orig_doc_id — ссылка на оригинальный документ, в случае, если данный документ является переводом;
- uploaded_by — ссылка на пользователя, загрузившего документ в систему;
- upload_date — дата загрузки документа в систему.

DocumentAuthor

Сущность DocumentAuthor формализует связь «многие-ко-многим» документов и авторов:

- doc_id — идентификатор документа;
- author_id — идентификатор автора документа.

Author

Сущность Author формализует автора документов:

- id — идентификатор автора;
- first_name — имя автора;
- last_name — фамилия автора;
- mid_name — отчество автора;
- email — электронная почта автора;
- creation_date — дата появления сущности в системе.

AnnotationTask

Сущность AnnotationTask формализует задание на разметку:

- id — идентификатор задания на разметку;
- orig_doc_id — ссылка на оригинальный документ, участвующий в разметке;
- trans_doc_id — ссылка на перевод документа, участвующий в разметке;
- description — описание задания на разметку;
- created_by — ссылка на пользователя, создавшего задание на разметку;
- creation_date — дата создания задания на разметку;
- last_update_date — дата последнего обновления задания на разметку.

StructAnnotation

Сущность StructAnnotation формализует структурную разметку:

- id — идентификатор структурной разметки;
- task_id — ссылка на задание на разметку, результатом выполнения которого данная разметка является;
- orig_doc_id — ссылка на документ, для которого выполняется структурная разметка;
- beg_sent_no — номер начального предложения разметки;
- end_sent_no — номер последнего предложения разметки;
- status — статус разметки (утверждена модератором или нет);
- done_by — ссылка на пользователя, выполнившего разметку.

TermAnnotation

Сущность TermAnnotation формализует терминологическую разметку:

- id — идентификатор терминологической разметки;
- task_id — ссылка на задание на разметку, результатом выполнения которого данная разметка является;
- orig_doc_id — ссылка на оригинальный документ, для которого выполняется терминологическая разметка;
- trans_doc_id — ссылка на перевод документа, для которого выполняется терминологическая разметка;
- status — статус разметки (утверждена модератором или нет);
- done_by — ссылка на пользователя, выполнившего разметку.

TermAnnotationPart

Сущность TermAnnotationPart формализует составляющие терминологической разметки — определяет границы терминов, входящих в разметку, и устанавливает их соответствие в оригинальном документе и его переводе:

- `annot_id` — ссылка на идентификатор терминологической разметки;
- `part_no` — номер части, термина, входящего в терминологическую разметку;
- `orig_sent_no` — номер предложения в оригинальном документе;
- `trans_sent_no` — номер предложения в документе-переводе;
- `beg_orig_token_no` — номер токена, являющегося началом термина в оригинальном документе;
- `end_orig_token_no` — номер токена, являющегося концом термина в оригинальном документе;
- `beg_trans_token_no` — номер токена, являющегося началом термина в документе-переводе;
- `end_trans_token_no` — номер токена, являющегося концом термина в документе-переводе.

Sentence

Сущность Sentence формализует предложение текста документа:

- `doc_id` — ссылка на документ, к которому относится предложение;
- `sent_no` — номер предложения в документе;
- `content` — текст предложения.

Token

Сущность Token формализует токен, получаемый в процессе токенизации текста документа:

- doc_id — ссылка на документ, в котором содержится предложение, которому принадлежит токен;
- sent_no — номер предложения в документе, которому принадлежит токен;
- token_no — номер токена в предложении;
- content — текст токена.

2.3 Описание ограничений целостности

В данном подразделе в таблицах 4 – 13 будут описаны проектируемые ограничения целостности базы данных в контексте выбранной, реляционной, модели.

Таблица 4 – Ограничение целостности User

Поле	Тип	Ограничение
id	UUID	первичный ключ
username	строка	уникальный, не пустое значение
email	строка	не пустое значение
pwd_hash	строка	не пустое значение
trust_level	вещественное число	не пустое значение, по умолчанию 0
creation_date	временная метка	не пустое значение
last_update_date	временная метка	не пустое значение

Таблица 5 – Ограничение целостности Document

Поле	Тип	Ограничение
id	UUID	первичный ключ
url	строка	не пустое значение
title	строка	не пустое значение
lang	строка	не пустое значение
orig_doc_id	UUID	внешний ключ на поле id таблицы Document
uploaded_by	UUID	внешний ключ на поле id таблицы User, не пустое значение
upload_date	временная метка	не пустое значение

Таблица 6 – Ограничение целостности DocumentAuthor

Поле	Тип	Ограничение
doc_id	UUID	первичный ключ, внешний ключ на поле id таблицы Document
author_id	UUID	первичный ключ, внешний ключ на поле id таблицы Author

Таблица 7 – Ограничение целостности Author

Поле	Тип	Ограничение
id	UUID	первичный ключ
first_name	строка	—
last_name	строка	—
mid_name	строка	—
email	строка	—
creation_date	временная метка	не пустое значение

Таблица 8 – Ограничение целостности AnnotationTask

Поле	Тип	Ограничение
id	UUID	первичный ключ
orig_doc_id	UUID	внешний ключ на поле id таблицы Document, не пустое значение
trans_doc_id	UUID	внешний ключ на поле id таблицы Document
description	строка	не пустое значение
created_by	UUID	внешний ключ на поле id таблицы User
creation_date	временная метка	не пустое значение
last_update_date	временная метка	не пустое значение

Таблица 9 – Ограничение целостности StructAnnotation

Поле	Тип	Ограничение
id	UUID	первичный ключ
task_id	UUID	внешний ключ на поле id таблицы AnnotationTask, не пустое значение
orig_doc_id	UUID	внешний ключ на поле id таблицы Document, не пустое значение
beg_sent_no	целое число	не пустое значение
end_sent_no	целое число	не пустое значение
status	строка	—
done_by	UUID	внешний ключ на поле id таблицы User, не пустое значение

Таблица 10 – Ограничение целостности TermAnnotation

Поле	Тип	Ограничение
id	UUID	первичный ключ
task_id	UUID	внешний ключ на поле id таблицы AnnotationTask, не пустое значение
orig_doc_id	UUID	внешний ключ на поле id таблицы Document, не пустое значение
trans_doc_id	UUID	внешний ключ на поле id таблицы Document
status	строка	—
done_by	UUID	внешний ключ на поле id таблицы User, не пустое значение

Таблица 11 – Ограничение целостности TermAnnotationPart

Поле	Тип	Ограничение
annot_id	UUID	первичный ключ, внешний ключ на поле id таблицы TermAnnotation
part_no	целое число	первичный ключ
orig_sent_no	целое число	не пустое значение
trans_sent_no	целое число	не пустое значение
beg_orig_token_no	целое число	не пустое значение
end_orig_token_no	целое число	не пустое значение
beg_trans_token_no	целое число	не пустое значение
end_trans_token_no	целое число	не пустое значение

Таблица 12 – Ограничение целостности Sentence

Поле	Тип	Ограничение
doc_id	UUID	первичный ключ, внешний ключ на поле id таблицы Document
sent_no	целое число	первичный ключ
content	строка	не пустое значение

Таблица 13 – Ограничение целостности Token

Поле	Тип	Ограничение
doc_id	UUID	первичный ключ, внешний ключ на поле id таблицы Document
sent_no	целое число	первичный ключ
token_no	целое число	первичный ключ
content	строка	не пустое значение

2.4 Описание функций, процедур и триггеров

Чем больше пользователь производит корректных разметок, тем сильнее модераторы могут его разметкам «доверять». Рейтинг доверия пользователей можно вычислять разными способами. Одним из примитивных способов вычисления рейтинга доверия пользователя может быть следующий: увеличивать рейтинг доверия пользователя на пять за каждую утвержденную разметку, и уменьшать на четыре — за каждую отклоненную. Пересчет рейтинга доверия пользователей можно производить автоматически, после проверки модератором его разметки, с помощью триггера. Алгоритм работы триггера, срабатывающего в ответ на обновление статуса разметки, приведен на рисунке 5 ниже.

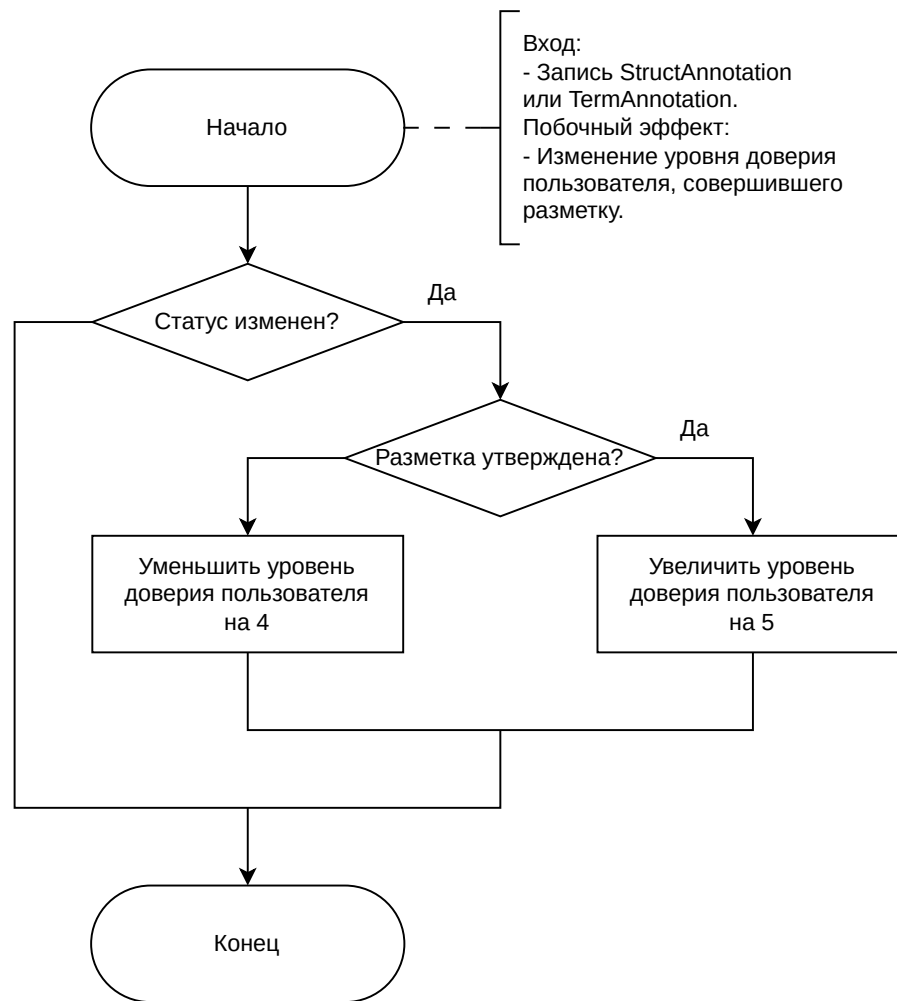


Рисунок 5 – Схема алгоритма работы триггера, отвечающего за пересчет уровня доверия пользователей

2.5 Описание ролевой модели

В разрабатываемой базе данных будет использоваться следующая ролевая модель:

- 1) Пользователь — имеет права доступа SELECT к таблицам Sentence, Token и AnnotationTask, права доступа INSERT к таблицам TermAnnotation, TermAnnotationPart и StructAnnotation.
- 2) Модератор — имеет все права доступа пользователя, а также INSERT к таблице AnnotationTask, UPDATE к таблицам AnnotationTask, TermAnnotation и StructAnnotation.
- 3) Администратор — имеет все права доступа ко всем таблицам.

Вывод

В данном разделе была представлена схема проектируемой базы данных, были описаны сущности базы данных, ограничения целостности, используемый триггер и используемая ролевая модель.

3 Технологический раздел

В данном разделе будет проведен выбор средств реализации базы данных и приложения к базе данных. Будут приведены описания реализаций сущностей базы данных, ограничений целостности и ролевой модели на уровне базы данных. Будет приведена схема реализованного триггера, продемонстрирована его работоспособность и описан интерфейс для доступа к базе данных.

3.1 Выбор средств реализации

В подразделах ниже будет проведен выбор средств для реализации базы данных и приложения к базе данных.

Выбор СУБД

Для реализации проектируемой базы данных подойдет любая реляционная СУБД с поддержкой ролевой модели. В настоящей работе была выбрана PostgreSQL [15] по следующим причинам:

- открытый исходный код;
- поддержка полнотекстового поиска [15];
- имеется опыт работы с данной СУБД.

Выбор языка программирования

Для реализации приложения к проектируемое БД был выбран язык программирования Go [16] по следующим причинам:

- простой синтаксис;
- быстрая компиляция [16];
- имеется стандартная библиотека для работы с SQL базами данных [17].

Выбор среды разработки

В качестве среды разработки был выбран Neovim [18] по следующим причинам:

- данный текстовый редактор позволяет редактировать файлы с исходным кодом программы;
- быстрая и удобная навигация по файлам проекта;
- есть опыт использования данного текстового редактора.

3.2 Описание реализаций

В подразделах ниже будет приведено описание реализаций сущностей базы данных, ограничений целостности базы данных, ролевой модели на уровне базы данных и реализованного триггера для корректировки уровня доверия пользователей.

3.2.1 Сущности базы данных

В листингах 1 – 10 приведены реализованные сущности базы данных.

```
1 CREATE TABLE IF NOT EXISTS user_data (  
2     id                                UUID,  
3     username                         VARCHAR(32) ,  
4     email                           VARCHAR(64) ,  
5     pwd_hash                         VARCHAR(256) ,  
6     trust_level                     FLOAT,  
7     creation_date                   TIMESTAMP,  
8     last_update_date                TIMESTAMP  
9 );
```

Листинг 1 – Создание User

```
1 CREATE TABLE IF NOT EXISTS document (  
2     id                                UUID,  
3     url                              VARCHAR(256) ,  
4     title                            VARCHAR(256) ,  
5     lang                             VARCHAR(2) ,  
6     orig_doc_id                      UUID,  
7     uploaded_by                      UUID,  
8     upload_date                      TIMESTAMP  
9 );
```

Листинг 2 – Создание Document

```
1 CREATE TABLE IF NOT EXISTS document_author (  
2     doc_id                            UUID,  
3     author_id                         UUID  
4 );
```

Листинг 3 – Создание DocumentAuthor

```
1 CREATE TABLE IF NOT EXISTS author (  
2     id                                UUID,  
3     first_name                        VARCHAR(64) ,  
4     last_name                         VARCHAR(64) ,  
5     mid_name                         VARCHAR(64) ,  
6     email                            VARCHAR(64) ,  
7     creation_date                     TIMESTAMP  
8 );
```

Листинг 4 – Создание Author

```

1 CREATE TABLE IF NOT EXISTS annotation_task (
2     id                                UUID,
3     orig_doc_id                       UUID,
4     trans_doc_id                      UUID,
5     description                       VARCHAR(256) ,
6     created_by                        UUID,
7     creation_date                     TIMESTAMP,
8     last_update_date                 TIMESTAMP
9 );

```

Листинг 5 – Создание AnnotationTask

```

1 CREATE TABLE IF NOT EXISTS struct_annotation (
2     id                                UUID,
3     task_id                           UUID,
4     orig_doc_id                       UUID,
5     beg_sent_no                       INT,
6     end_sent_no                       INT,
7     status                            VARCHAR(64) ,
8     done_by                           UUID
9 );

```

Листинг 6 – Создание StructAnnotation

```

1 CREATE TABLE IF NOT EXISTS term_annotation (
2     id                                UUID,
3     task_id                           UUID,
4     orig_doc_id                       UUID,
5     trans_doc_id                      UUID,
6     status                            VARCHAR(64) ,
7     done_by                           UUID
8 );

```

Листинг 7 – Создание TermAnnotation

```

1 CREATE TABLE IF NOT EXISTS term_annotation_part (
2     annot_id          UUID,
3     part_no           INT,
4     orig_sent_no      INT,
5     trans_sent_no     INT,
6     beg_orig_token_no INT,
7     end_orig_token_no INT,
8     beg_trans_token_no INT,
9     end_trans_token_no INT
10 );

```

Листинг 8 – Создание TermAnnotationPart

```

1 CREATE TABLE IF NOT EXISTS sentence (
2     doc_id          UUID,
3     sent_no         INT,
4     content         VARCHAR(256)
5 );

```

Листинг 9 – Создание Sentence

```

1 CREATE TABLE IF NOT EXISTS token (
2     doc_id          UUID,
3     sent_no         INT,
4     token_no        INT,
5     content         VARCHAR(256)
6 );

```

Листинг 10 – Создание Token

3.2.2 Ограничения целостности базы данных

В листингах 11 – 20 приведены реализованные ограничения целостности базы данных.

```
1 ALTER TABLE user_data
2     ALTER COLUMN id SET NOT NULL,
3     ADD CONSTRAINT pk_user_data_id
4     PRIMARY KEY (id),
5     ALTER COLUMN username SET NOT NULL,
6     ADD CONSTRAINT unique_user_data_username
7     UNIQUE (username),
8     ALTER COLUMN email SET NOT NULL,
9     ALTER COLUMN pwd_hash SET NOT NULL,
10    ALTER COLUMN trust_level SET NOT NULL,
11    ALTER COLUMN trust_level SET DEFAULT 0,
12    ALTER COLUMN creation_date SET NOT NULL,
13    ALTER COLUMN last_update_date SET DEFAULT NULL;
```

Листинг 11 – Ограничения User

```
1 ALTER TABLE document
2     ALTER COLUMN id SET NOT NULL,
3     ADD CONSTRAINT pk_document_id
4     PRIMARY KEY (id),
5     ALTER COLUMN url SET NOT NULL,
6     ALTER COLUMN title SET NOT NULL,
7     ALTER COLUMN lang SET NOT NULL,
8     ADD CONSTRAINT fk_document_orig_doc_id
9     FOREIGN KEY (orig_doc_id)
10    REFERENCES document(id),
11    ALTER COLUMN uploaded_by SET NOT NULL,
12    ADD CONSTRAINT fk_document_uploaded_by
13    FOREIGN KEY (uploaded_by)
14    REFERENCES user_data(id),
15    ALTER COLUMN upload_date SET NOT NULL;
```

Листинг 12 – Ограничения Document

```
1 ALTER TABLE document_author
2     ALTER COLUMN doc_id SET NOT NULL,
3     ADD CONSTRAINT fk_document_author_doc_id
4     FOREIGN KEY (doc_id)
5     REFERENCES document(id)
6     ON DELETE CASCADE,
7     ALTER COLUMN author_id SET NOT NULL,
8     ADD CONSTRAINT fk_document_author_author_id
9     FOREIGN KEY (author_id)
10    REFERENCES author(id);
```

Листинг 13 – Ограничения DocumentAuthor

```
1 ALTER TABLE author
2     ALTER COLUMN id SET NOT NULL,
3     ADD CONSTRAINT pk_author_id
4     PRIMARY KEY (id),
5     ALTER COLUMN creation_date SET NOT NULL;
```

Листинг 14 – Ограничения Author


```

1 ALTER TABLE annotation_task
2     ALTER COLUMN id SET NOT NULL,
3     ADD CONSTRAINT pk_annotation_task_id
4     PRIMARY KEY (id),
5     ALTER COLUMN orig_doc_id SET NOT NULL,
6     ADD CONSTRAINT fk_annotation_task_orig_doc_id
7     FOREIGN KEY (orig_doc_id)
8     REFERENCES document(id),
9     ADD CONSTRAINT fk_annotation_task_trans_doc_id
10    FOREIGN KEY (trans_doc_id)
11    REFERENCES document(id),
12    ALTER COLUMN description SET NOT NULL,
13    ADD CONSTRAINT fk_annotation_task_created_by
14    FOREIGN KEY (created_by)
15    REFERENCES user_data(id),
16    ALTER COLUMN creation_date SET NOT NULL,
17    ALTER COLUMN last_update_date SET NOT NULL;

```

Листинг 15 – Ограничения AnnotationTask

```

1 ALTER TABLE struct_annotation
2     ALTER COLUMN id SET NOT NULL,
3     ADD CONSTRAINT pk_struct_annotation_id
4     PRIMARY KEY (id),
5     ALTER COLUMN task_id SET NOT NULL,
6     ADD CONSTRAINT fk_struct_annotation_task_id
7     FOREIGN KEY (task_id)
8     REFERENCES annotation_task(id),
9     ALTER COLUMN orig_doc_id SET NOT NULL,
10    ADD CONSTRAINT fk_struct_annotation_orig_doc_id
11    FOREIGN KEY (orig_doc_id)
12    REFERENCES document(id),
13    ALTER COLUMN beg_sent_no SET NOT NULL,
14    ALTER COLUMN end_sent_no SET NOT NULL,
15    ALTER COLUMN done_by SET NOT NULL,
16    ADD CONSTRAINT fk_struct_annotation_done_by
17    FOREIGN KEY (done_by)
18    REFERENCES user_data(id);

```

Листинг 16 – Ограничения StructAnnotation

```

1 ALTER TABLE term_annotation
2     ALTER COLUMN id SET NOT NULL,
3     ADD CONSTRAINT pk_term_annotation_id
4     PRIMARY KEY (id),
5     ALTER COLUMN task_id SET NOT NULL,
6     ADD CONSTRAINT fk_term_annotation_task_id
7     FOREIGN KEY (task_id)
8     REFERENCES annotation_task(id),
9     ALTER COLUMN orig_doc_id SET NOT NULL,
10    ADD CONSTRAINT fk_term_annotation_orig_doc_id
11    FOREIGN KEY (orig_doc_id)
12    REFERENCES document(id),
13    ADD CONSTRAINT fk_term_annotation_trans_doc_id
14    FOREIGN KEY (trans_doc_id)
15    REFERENCES document(id),
16    ALTER COLUMN done_by SET NOT NULL,
17    ADD CONSTRAINT fk_term_annotation_done_by
18    FOREIGN KEY (done_by)
19    REFERENCES user_data(id);

```

Листинг 17 – Ограничения TermAnnotation

```

1 ALTER TABLE term_annotation_part
2     ADD CONSTRAINT pk_term_annotation_part_annot_id_part_no
3     PRIMARY KEY (annot_id, part_no),
4     ADD CONSTRAINT fk_term_annotation_part_annot_id
5     FOREIGN KEY (annot_id)
6     REFERENCES term_annotation(id)
7     ON DELETE CASCADE,
8     ALTER COLUMN orig_sent_no SET NOT NULL,
9     ALTER COLUMN trans_sent_no SET NOT NULL,
10    ALTER COLUMN beg_orig_token_no SET NOT NULL,
11    ALTER COLUMN end_orig_token_no SET NOT NULL,
12    ALTER COLUMN beg_trans_token_no SET NOT NULL,
13    ALTER COLUMN end_trans_token_no SET NOT NULL;

```

Листинг 18 – Ограничения TermAnnotationPart

```
1 ALTER TABLE sentence
2     ADD CONSTRAINT pk_sentence_doc_id_sent_no
3     PRIMARY KEY (doc_id, sent_no),
4     ADD CONSTRAINT fk_sentence_doc_id
5     FOREIGN KEY (doc_id)
6     REFERENCES document(id),
7     ALTER COLUMN content SET NOT NULL;
```

Листинг 19 – Ограничения Sentence

```
1 ALTER TABLE token
2     ADD CONSTRAINT pk_token_doc_id_sent_no_token_no
3     PRIMARY KEY (doc_id, sent_no, token_no),
4     ADD CONSTRAINT fk_token_doc_id
5     FOREIGN KEY (doc_id)
6     REFERENCES document(id),
7     ALTER COLUMN content SET NOT NULL;
```

Листинг 20 – Ограничения Token

3.2.3 Ролевая модель на уровне базы данных

В листинге 21 приведена реализованная ролевая модель на уровне базы данных.

```
1 CREATE ROLE user_role ;
2 GRANT INSERT ON TABLE term_annotation TO user_role ;
3 GRANT INSERT ON TABLE term_annotation_part TO user_role ;
4 GRANT INSERT ON TABLE struct_annotation TO user_role ;
5 GRANT SELECT ON TABLE annotation_task TO user_role ;
6 GRANT SELECT ON TABLE sentence TO user_role ;
7 GRANT SELECT ON TABLE token TO user_role ;
8
9 CREATE ROLE moderator_role ;
10 GRANT user_role TO moderator_role ;
11 GRANT INSERT ON TABLE annotation_task TO moderator_role ;
12 GRANT UPDATE ON TABLE annotation_task TO moderator_role ;
13 GRANT UPDATE ON TABLE term_annotation TO moderator_role ;
14 GRANT UPDATE ON TABLE struct_annotation TO moderator_role ;
15
16 CREATE ROLE admin_role WITH SUPERUSER ;
17 GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin_role ;
18
19 CREATE USER dbusr WITH PASSWORD 'dbusr' ;
20 CREATE USER moder WITH PASSWORD 'moder' ;
21 CREATE USER admin WITH PASSWORD 'admin' ;
22
23 GRANT user_role TO dbusr ;
24 GRANT moderator_role TO moder ;
25 GRANT admin_role TO admin ;
```

Листинг 21 – Создание ролей и пользователей

3.2.4 Функции, процедуры и триггеры

На рисунке 6 представлена реализация функции триггера, отвечающего за пересчет уровня доверия пользователей.

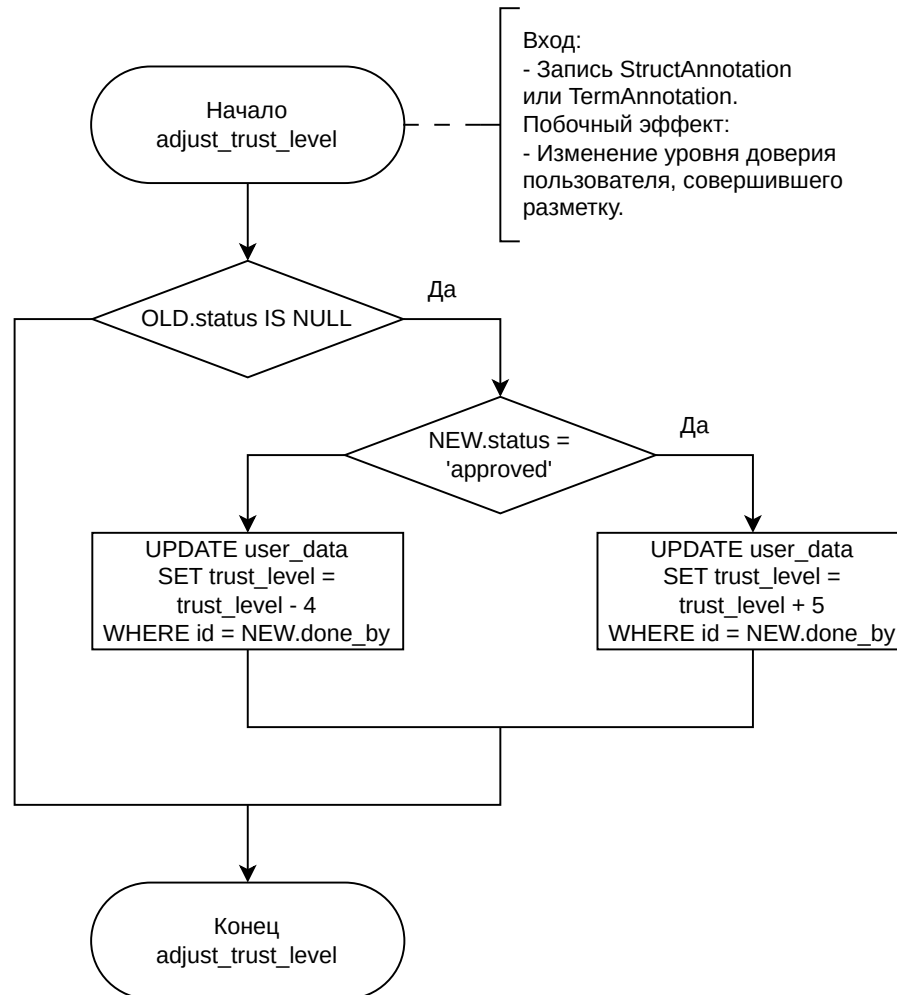


Рисунок 6 – Схема реализации алгоритма работы триггера, отвечающего за пересчет уровня доверия пользователей на языке plpgsql

3.3 Тестирование функций

В листингах 22, 23 продемонстрирована работоспособность реализованного триггера.

```

1 db=> select username, trust_level, status from user_data u join
      struct_annotation s on u.id = s.done_by where s.id =
      'b72d276e-a6ca-402b-bcdb-480da5acb9dd';
2
3 username | trust_level | status
4 -----+-----+-----
5 rgallegos988 | 0 |
6
7 db=> update struct_annotation set status = 'approved' where id =
      'b72d276e-a6ca-402b-bcdb-480da5acb9dd';
8
9 UPDATE 1
10 db=> select username, trust_level, status from user_data u join
      struct_annotation s on u.id = s.done_by where s.id =
      'b72d276e-a6ca-402b-bcdb-480da5acb9dd';
11
12 username | trust_level | status
13 -----+-----+-----
14 rgallegos988 | 5 | approved

```

Листинг 22 – Увеличение уровня доверия пользователя при одобрении его разметки

```

1 db=> select username, trust_level, status from user_data u join
      struct_annotation s on u.id = s.done_by where s.id =
      '595c15b7-a289-4f3c-8069-4c678baaa8ad';
2
3 username | trust_level | status
4 -----+-----+-----
5 christopher097599 | 0 |
6
7 db=> update struct_annotation set status = 'rejected' where id =
      '595c15b7-a289-4f3c-8069-4c678baaa8ad';
8
9 UPDATE 1
10 db=> select username, trust_level, status from user_data u join
      struct_annotation s on u.id = s.done_by where s.id =
      '595c15b7-a289-4f3c-8069-4c678baaa8ad';
11
12 username | trust_level | status
13 -----+-----+-----
14 christopher097599 | -4 | rejected

```

Листинг 23 – Уменьшение уровня доверия пользователя при отклонении его разметки

3.4 Интерфейс доступа к базе данных

В таблице ниже представлен интерфейс доступа к базе данных, осуществляемого с помощью REST API [19].

Таблица 14 – Интерфейс доступа к базе данных

Метод	URL	Описание
POST	/register	Регистрация пользователя с введенными именем пользователя, адресом почты и паролем
POST	/login	Вход в систему по имени пользователя и паролю
GET	/d/{id}	Получение текста документа с указанным id
POST	/d	Добавление нового документа
GET	/tasks	Получение списка заданий для разметки
GET	/task/{id}	Получение страницы с конкретной задачей на разметку
POST	/task	Создание нового задания на разметку
GET	/a/{id}	Получение разметки с указанным id
POST	/a	Добавление новой разметки
GET	/search	Получение страницы для поиска по корпусу
POST	/search	Выполнение поиска по корпусу

Вывод

В данном разделе был проведен выбор средств реализации базы данных и приложения к базе данных. Были приведены описания реализаций сущностей базы данных, ограничений целостности и ролевой модели на уровне базы данных. Была приведена схема реализованного триггера, продемонстрирована его работоспособность и описан интерфейс для доступа к базе данных.

4 Исследовательский раздел

В данном разделе будет приведено описание исследования зависимости времени ответа от количества запросов в секунду, технические характеристики устройства, на котором оно проводилось, и представлены его результаты.

Цель исследования

Целью исследования является получение зависимости времени ответа от количества запросов в секунду и сравнения эффективности реализаций приложения с использованием дополнительного кеширования и без него.

4.1 Описание исследования

В качестве базы данных для дополнительного кеширования была выбрана KeyDB [20], так как ее функциональности достаточно для обеспечения кеширования, а также она является бесплатной и открытой альтернативой Redis [21] — известной базы данных для этих целей [22].

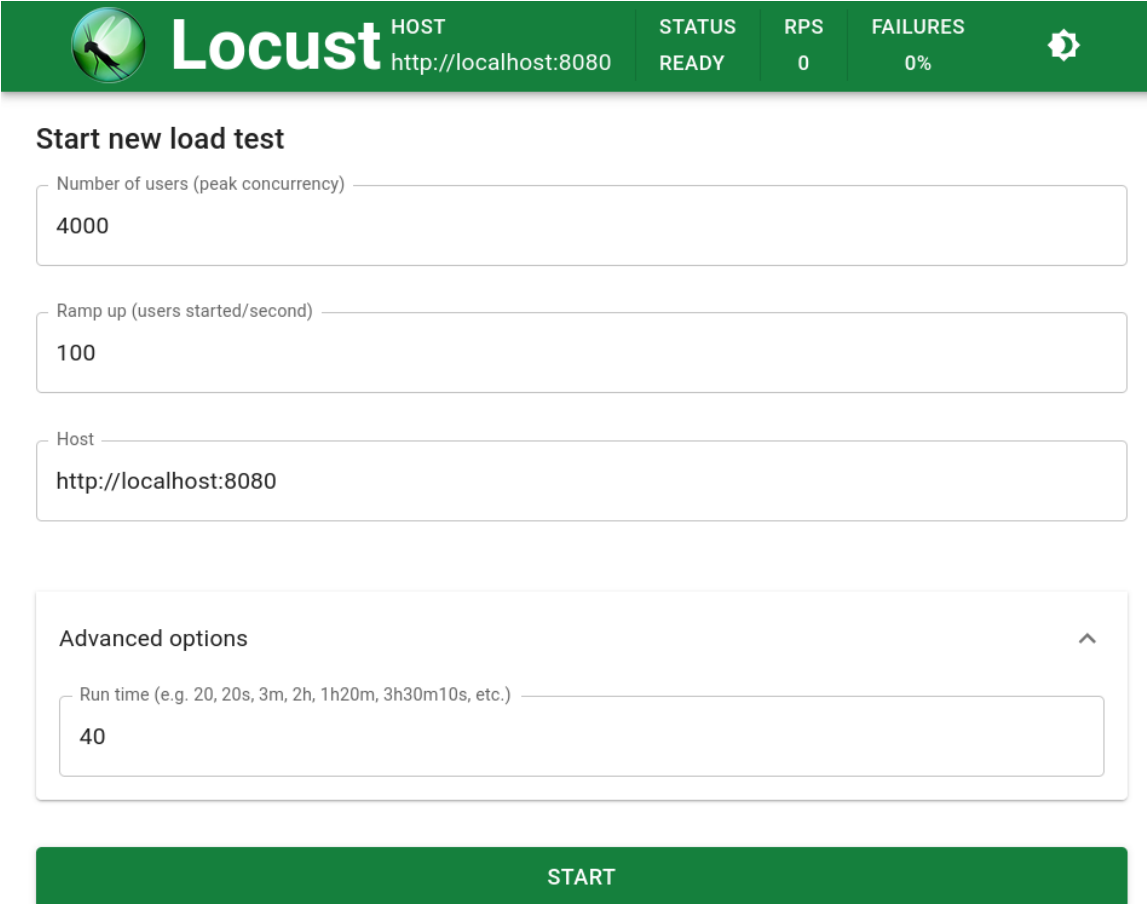
В качестве инструмента для проведения нагрузочного тестирования был выбран Locust [23], так как его функциональности достаточно для проведения нагрузочного тестирования, а также присутствует возможность написания тестовых сценариев на простом скриптовом языке Python [24].

Проведение нагрузочного тестирования в Locust состоит из нескольких этапов:

- 1) Написание сценариев тестирования в файле `locustfile.py`;
- 2) Установка максимального количества запросов в секунду, шага увеличения количества запросов в секунду и времени, на протяжении которого будет проводиться нагрузочного тестирования;
- 3) Просмотр и экспорт статистики и результатов проведенного нагрузочного тестирования.

Содержимое файла со сценариями тестирования приведено в листинге 25 приложения А; пример использования кеша в приложении — в листинге

24, а примеры интерфейса Locust, соответствующие этапам 2 и 3 проведения нагрузочного тестирования — на рисунках 7 – 9 ниже.



The screenshot shows the Locust web interface. At the top is a green header bar with the Locust logo (a green circle with a black locust) on the left. To its right is the text "Locust" in large white font, followed by "HOST" and "http://localhost:8080" in smaller white font. Further right are three white boxes with green borders: "STATUS" containing "READY", "RPS" containing "0", and "FAILURES" containing "0%". On the far right of the header is a white gear icon.

Below the header, the section "Start new load test" is displayed. It contains three input fields with labels and values:

- Label: "Number of users (peak concurrency)" — Value: "4000"
- Label: "Ramp up (users started/second)" — Value: "100"
- Label: "Host" — Value: "http://localhost:8080"

Below these fields is a section titled "Advanced options" with a small upward arrow icon on the right. It contains one input field with label "Run time (e.g. 20, 20s, 3m, 2h, 1h20m, 3h30m10s, etc.)" and value "40".

At the bottom of the interface is a large green button with the text "START" in white capital letters.

Рисунок 7 – Запуск нагрузочного тестирования в Locust

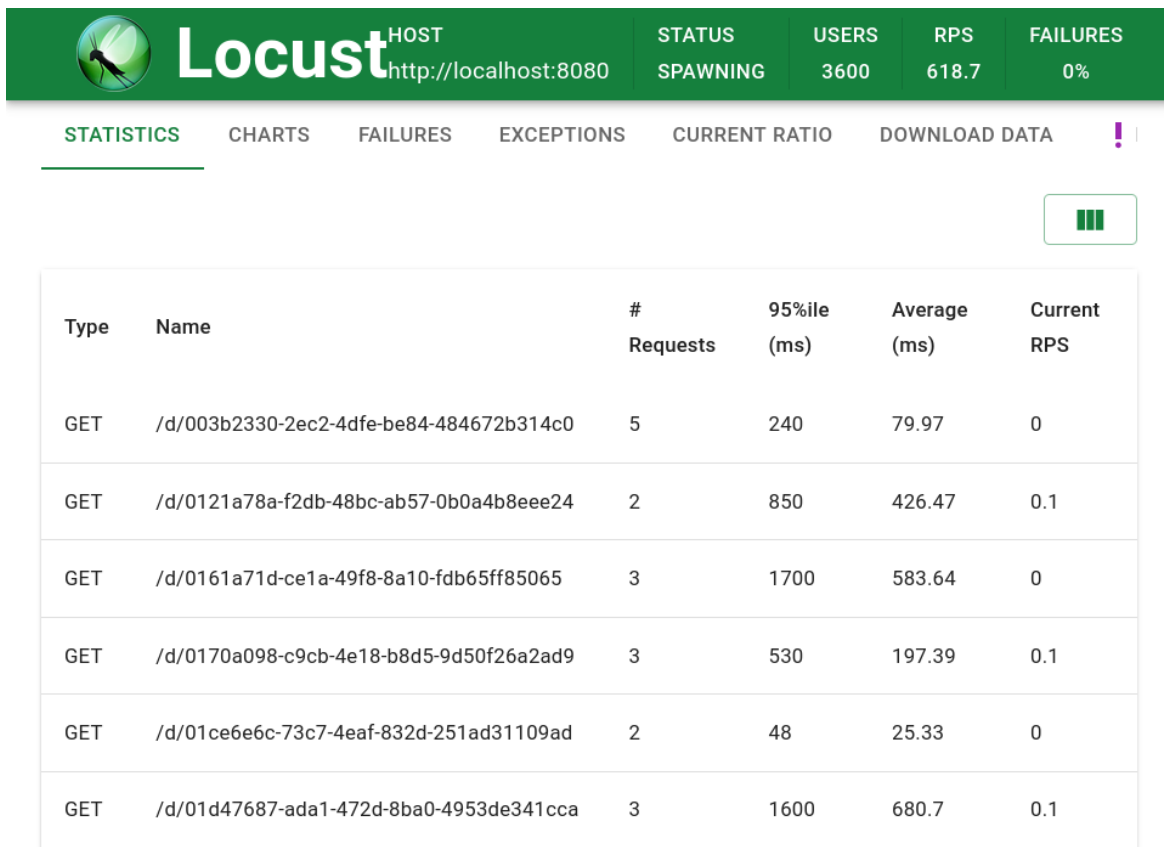


Рисунок 8 – Просмотр статистики нагрузочного тестирования в реальном времени

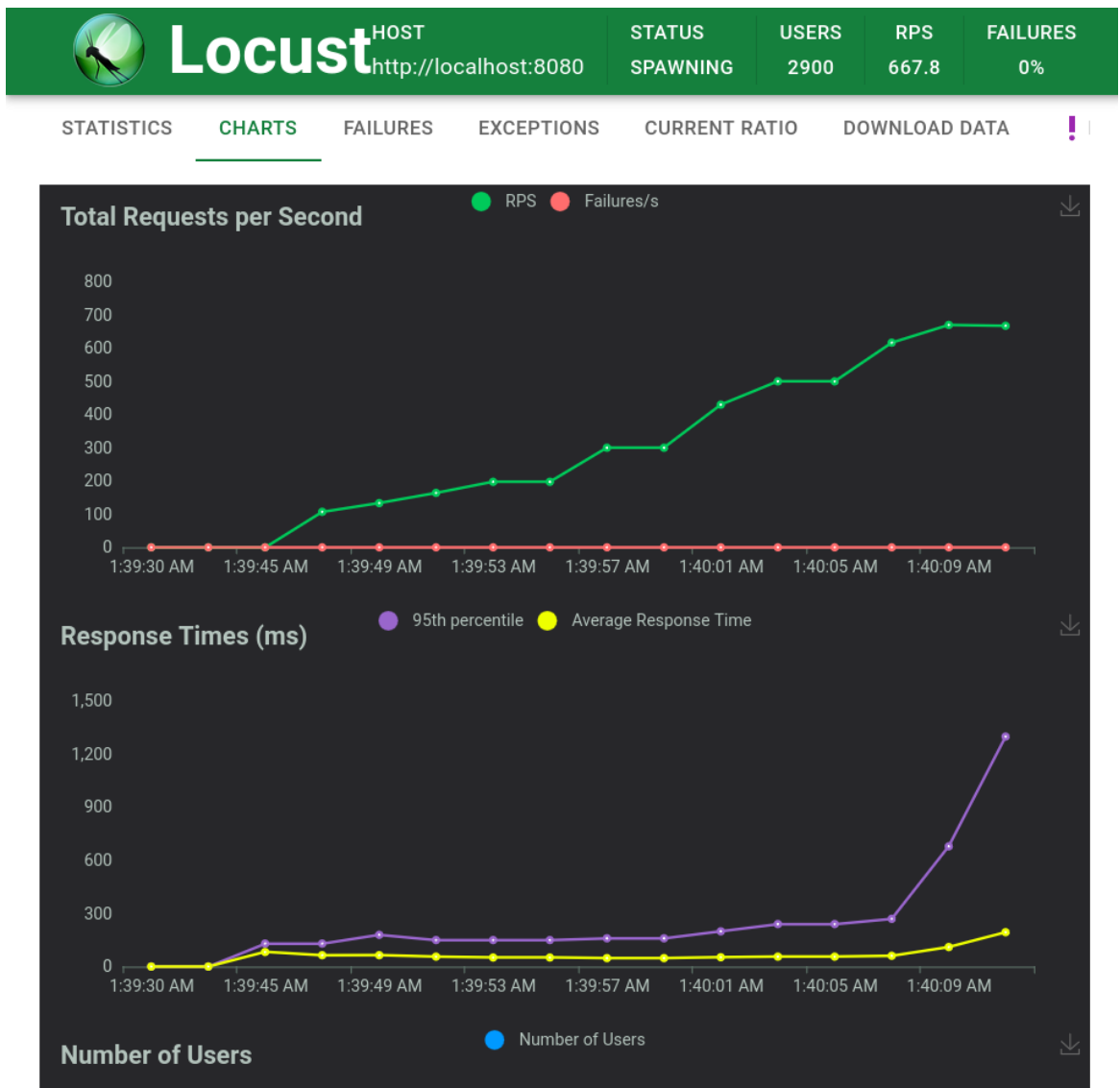


Рисунок 9 – Просмотр графиков нагрузочного тестирования в реальном времени

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры времени, представлены ниже.

- 1) Процессор: AMD Ryzen 7 4700U 2.0 ГГц [25], 8 физических ядер, 8 потоков;
- 2) Оперативная память: 8 ГБ, DDR4, 3200 МГц;
- 3) Операционная система: NixOS 23.11.1209.933d7 [26];
- 4) Версия ядра: 6.1.64.

При выполнении замеров времени ноутбук был подключен к сети электропитания, был запущен браузер Chromium [27] с одной вкладкой, три терминала Alacritty [28], в которых выполнялись Locust, сервер на Go и Neovim. На фоне работал Docker-контейнер [29] с базой данных.

4.3 Результаты исследования

Далее приведены таблицы и графики с результатами исследования. Поясняющие комментарии к таблицам приводятся после таблицы 18; к графикам — после каждого графика.

Данные в правых колонках таблиц являются усредненным значением из общего количества запросов к моменту времени, когда число запросов в секунду составляло значение из левой колонки таблицы. Так, например, в таблице 15 время ответа 0.086 секунд является средним из 10 запросов, а время ответа 0.555 секунд — средним из 15438 запросов, так как нагрузочное тестирование проводилось на протяжении 40 секунд с приростом количества имитируемых пользователей приложения — 100 новых пользователей в секунду. В таблице 17 время ответа 0.465 является средним из 16103 запросов.

Таблица 15 – Зависимость среднего времени ответа от количества запросов в секунду без использования кеширования

Число запросов в секунду	Среднее время ответа (с)
10	0.086
127	0.068
161	0.058
193	0.060
225	0.050
387	0.044
456	0.047
581	0.055
644	0.070
696	0.095
722	0.222
727	0.290
741	0.465
752	0.555

Таблица 16 – Зависимость 95-го перцентиля времени ответа от количества запросов в секунду без использования кеширования

Число запросов в секунду	95-й перцентиль времени ответа (с)
10	0.110
127	0.160
161	0.150
193	0.180
225	0.160
387	0.160
456	0.170
581	0.260
644	0.330
696	0.450
722	1.200
727	1.600
741	2.800
752	3.800

Таблица 17 – Зависимость среднего времени ответа от количества запросов в секунду при использовании кеширования

Число запросов в секунду	Среднее время ответа (с)
10	0.086
112	0.070
172	0.063
202	0.060
236	0.053
300	0.050
362	0.049
424	0.050
493	0.054
611	0.058
685	0.074
730	0.110
761	0.238
771	0.303
780	0.465

Таблица 18 – Зависимость 95-го перцентиля времени ответа от количества запросов в секунду при использовании кеширования

Число запросов в секунду	95-й перцентиль времени ответа (с)
10	0.130
112	0.120
172	0.170
202	0.180
236	0.170
300	0.170
362	0.180
424	0.180
493	0.210
611	0.250
685	0.320
730	0.470
761	1.300
771	1.500
780	3.100

По таблицам 15 и 17 можно увидеть, что при 752 запросах в секунду к приложению, не использующему кеш KeyDB, среднее время ответа составляет 0.555 секунд, а при 780 запросах в секунду к приложению, использующему кеш KeyDB, среднее время ответа составляет 0.465 секунд.

По таблицам 16 и 18 можно увидеть, что 95 процентов запросов при 752 запросах в секунду к приложению, не использующему кеш KeyDB, обрабатываются быстрее 3.8 секунд, а при 780 запросах в секунду к приложению, использующему кеш KeyDB, 95 процентов запросов обрабатываются быстрее 3.1 секунды.

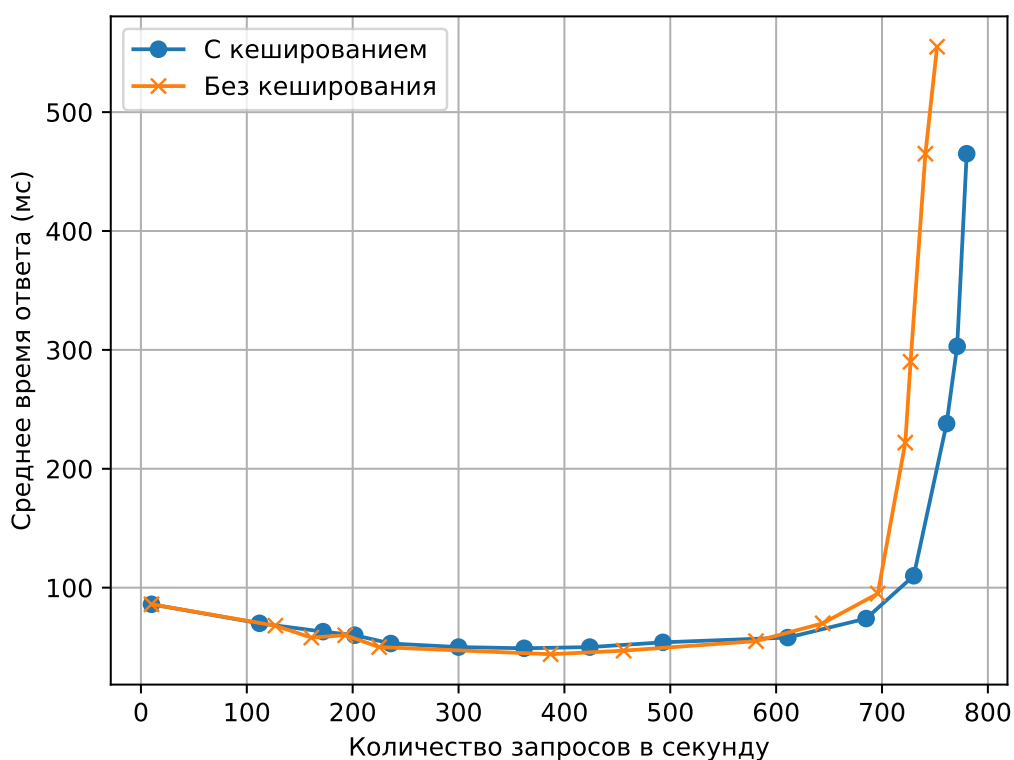


Рисунок 10 – Зависимость среднего времени ответа от количества запросов в секунду

По графику 10 можно заметить, что среднее время ответа убывает примерно до 350 запросов в секунду, после чего начинается экспоненциальный рост. Более высокие значения среднего времени ответа на ранних этапах обусловлены тем, что большая часть запрашиваемых данных пока еще не находится в кеше KeyDB или базы данных. После 600 запросов в секунду использование кеша KeyDB начинает давать прирост производительности. При 730 запросах в секунду запросы к приложению, использующему кеш KeyDB, обрабатываются быстрее примерно в 2.64 раза.

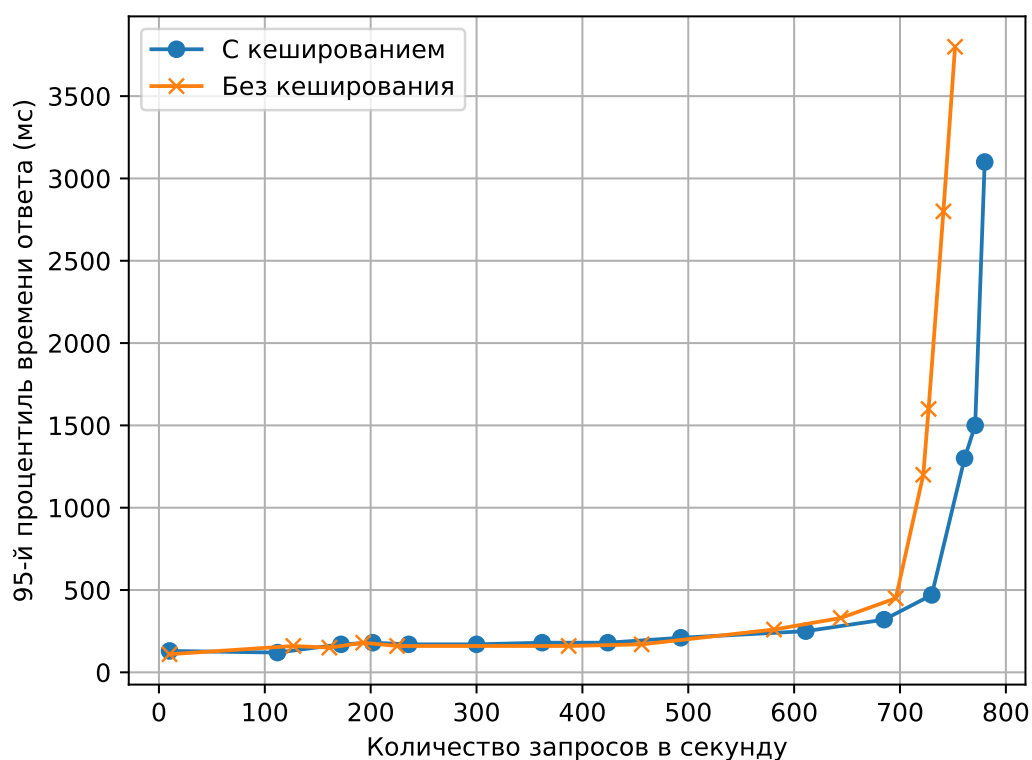


Рисунок 11 – Зависимость 95-го перцентиля времени ответа от количества запросов в секунду

По графику 11 видно, что при 750 запросах в секунду 95 процентов запросов к приложению, использующему кеш KeyDB, обрабатываются примерно в 3.45 раза быстрее. После 700 запросов в секунду возникает резкий скачок 95-го перцентиля времени ответа и начинается экспоненциальный рост.

Вывод

В данном разделе было проведено исследование зависимости времени ответа от количества запросов в секунду. Среднее время ответа увеличивается с увеличением количества запросов в секунду. В определенный момент среднее время ответа возрастает в 1.91 раз с увеличением количества запросов в секунду в 1.05 раза.

Использование дополнительного кеша позволяет ускорить среднее время ответа. Так при 752 запросах в секунду использование кеша позволило ускорить среднее время ответа более, чем в 3 раза.

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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.
2. Гаврилова Ю.М. Конспект лекций по курсу «Базы Данных». 2023.
3. Бутенко Ю.И., Сулейманова Э.В. Терминологическая разметка научно-технических текстов в специальном корпусе // Проблемы лингвистики и лингводидактики в неязыковом вузе: 5-я Международная научно-практическая конференция. 2022. Т. 1. С. 329–337.
4. Захаров В.П., Богданова С.Ю. Корпусная лингвистика: учебник. 3-е изд., перераб. — СПб.: Изд-во С.-Петербур. ун-та, 2020. — 234 с.
5. Лесников В.С. Виды разметок текстовых корпусов русского языка // Научно-техническая информация. Сер. 2. Информационные процессы и системы. 2019. № 9. С. 27–30.
6. Бутенко Ю.И. Модель учебно-научного текста для разметки корпуса научно-технических текстов // Экономика. Информатика. 2021. Т. 48, № 1. С. 123–129.
7. Бутенко Ю.И., Попова Н.М. Особенности семантической разметки в корпусе научно-технических текстов // Проблемы лингвистики и лингводидактики в неязыковом вузе: 5-я Международная научно-практическая конференция. 2022. Т. 1. С. 324–328.
8. University Centre for Computer Corpus Research on Language [Электронный ресурс]. — URL: <https://ucrel.lancs.ac.uk/usas> (дата обращения: 20.05.2024).
9. Бутенко Ю.И., Киселёва А.Д. Анализ современных корпусов параллельных текстов // Актуальные проблемы лингвистики и лингводидактики в неязыковом вузе: 4-я Международная научно-практическая конференция. 2020. Т. 1. С. 238–242.

10. Sketch Engine: Create and search a text corpus [Электронный ресурс]. – URL: <https://www.sketchengine.eu> (дата обращения: 20.05.2024).
11. NoSketch Engine [Электронный ресурс]. – URL: <https://nlp.fi.muni.cz/trac/noske> (дата обращения: 20.05.2024).
12. OpenCorpora — открытый корпус [Электронный ресурс]. – URL: <https://opencorpora.org> (дата обращения: 25.03.2024).
13. Бутенко Ю.И., Строганов Ю.В., Бабаджанян Р.В. Исследовательский прототип параллельного корпуса научно-технических текстов // Актуальные проблемы лингвистики и лингводидактики в неязыковом вузе: 4-я Международная научно-практическая конференция. 2020. Т. 1. С. 205–209.
14. Date C. J. Relational Database Writings, 1991-1994. — Addison-Wesley Publishing Company., 1995 — 542 с.
15. PostgreSQL: The World’s Most Advanced Open Source Relational Database [Электронный ресурс]. – URL: <https://www.postgresql.org/about> (дата обращения: 01.06.2024).
16. Build simple, secure, scalable systems with Go [Электронный ресурс]. – URL: <https://go.dev> (дата обращения: 01.06.2024).
17. Go — sql package [Электронный ресурс]. – URL: <https://pkg.go.dev/database/sql> (дата обращения: 01.06.2024).
18. Neovim [Электронный ресурс]. – URL: <https://neovim.io> (дата обращения: 01.06.2024).
19. What is a REST API? [Электронный ресурс]. – URL: <https://www.ibm.com/topics/rest-apis> (дата обращения: 03.06.2024).
20. KeyDB — The Faster Redis Alternative [Электронный ресурс]. – URL: <https://docs.keydb.dev> (дата обращения: 06.06.2024).
21. Redis — The Real-time Data Platform [Электронный ресурс]. – URL: <https://redis.io> (дата обращения: 06.06.2024).

22. Jobs in UK citing Redis [Электронный ресурс]. – URL: <https://www.itjobswatch.co.uk/jobs/uk/redis.do> (дата обращения: 06.06.2024).
23. LOCUST — An open source load testing tool [Электронный ресурс]. – URL: <https://locust.io> (дата обращения: 03.06.2024).
24. Python — A programming language that lets you work quickly and integrate systems more effectively [Электронный ресурс]. – URL: <https://www.python.org> (дата обращения: 06.06.2024).
25. AMD Ryzen 7 4700U [Электронный ресурс]. – URL: <https://www.amd.com/en/product/9096> (дата обращения: 03.06.2024).
26. NixOS [Электронный ресурс]. – URL: <https://nixos.org> (дата обращения: 03.06.2024).
27. The Chromium Projects — Chromium [Электронный ресурс]. – URL: <https://www.chromium.org/Home> (дата обращения: 03.06.2024).
28. Alacritty — A fast, cross-platform, OpenGL terminal emulator [Электронный ресурс]. – URL: <https://github.com/alacritty/alacritty> (дата обращения: 03.06.2024).
29. What is Docker? — Accelerate how you build, share, and run applications [Электронный ресурс]. – URL: <https://www.docker.com> (дата обращения: 03.06.2024).

ПРИЛОЖЕНИЕ А

```
1 type DocumentHandler struct {
2     service *s.DocumentService
3     cache   *keydb.Client
4     store   *sessions.CookieStore
5 }
6
7 // ...
8
9 func (h *DocumentHandler) GetDocumentText(c echo.Context) error {
10     documentID := c.Param("id")
11     useCache := true
12
13     if useCache {
14         cachedText, err := h.cache.Get(ctx, documentID).Result()
15         if err == keydb.Nil {
16             uid, err := uuid.Parse(documentID)
17             if err != nil {
18                 return c.String(http.StatusBadRequest, "Invalid_
19                     document_ID_-_parse_uuid")
20             }
21             text, err := h.service.GetDocumentText(uid)
22             if err != nil {
23                 return c.String(http.StatusInternalServerError,
24                     err.Error())
25             }
26             err = h.cache.Set(ctx, documentID, text,
27                 10*time.Minute).Err()
28             if err != nil {
29                 return c.String(http.StatusInternalServerError,
30                     "Failed_to_cache_document_text")
31             }
32             return c.String(http.StatusOK, text)
33         } else if err != nil {
34             return c.String(http.StatusBadRequest, "Invalid_
35                 document_ID")
36         }
37     }
```

```

35         return c.String(http.StatusOK, cachedText)
36     } else {
37         uid, err := uuid.Parse(documentID)
38         if err != nil {
39             return c.String(http.StatusBadRequest, "Invalid_
               document_ID_-_parse_uuid")
40         }
41
42         text, err := h.service.GetDocumentText(uid)
43         if err != nil {
44             return c.String(http.StatusInternalServerError,
               err.Error())
45         }
46
47         return c.String(http.StatusOK, text)
48     }
49 }

```

Листинг 24 – Кеширование текста документов


```

1 from faker import Faker
2 from locust import HttpUser, TaskSet, task, between
3
4 fake = Faker()
5
6 words = ["the", "be", "of", "and", "a", "to", "in", "he", ...]
7 document_ids = ["41b9023a-77d4-4d4e-aa2c-8305b937443a", ...]
8
9 class UserBehavior(TaskSet):
10     @task(2) # Вес задачи - 2
11     def post_search(self):
12         headers = {"Content-Type": "application/json"}
13         payload = {
14             "content": fake.random_element(words),
15         }
16         self.client.post("/search", json=payload, headers=headers)
17
18     @task(1)
19     def get_register(self):
20         self.client.get("/register")
21     # ...
22
23     @task(1)
24     def get_search(self):
25         self.client.get("/search")
26
27     @task(1)
28     def get_document(self):
29         id = fake.random_element(document_ids)
30         self.client.get(f"/d/{id}")
31
32 class WebsiteUser(HttpUser):
33     tasks = [UserBehavior]
34     wait_time = between(1, 5)

```

Листинг 25 – Содержимое файла locustfile.py для проведения нагрузочного тестирования

ПРИЛОЖЕНИЕ Б

Презентация к курсовой работе

Презентация состоит из N слайдов.