



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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка базы данных для хранения и обработки
результатов проведения тестов Тьюринга

Студент ИУ7-65Б
(Группа)

С. К. Романов
(Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта

К.А. Кивва
(Подпись, дата) (И.О.Фамилия)

Консультант

К.А. Кивва
(Подпись, дата) (И.О.Фамилия)

2023 г.

РЕФЕРАТ

Расчётно-пояснительная записка содержит 59 с., 16 рис., 3 табл., 31 ист.

Цель работы: создание базы данных для хранения результатов тестов Тьюринга.

Ключевые слова: базы данных, SurrealDB, граф, отношения.

В данной работе проводится изучение принципов работы баз данных.

Объектом исследования является модель представления результатов тестов Тьюринга в виде графа.

Результаты: разработана программа, предназначенная для работы с базами данных. Проанализированы разные системы управления базами данных.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ОПРЕДЕЛЕНИЯ	5
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	7
ВВЕДЕНИЕ	8
1 Аналитический раздел	10
1.1 Анализ предметной области	10
1.2 Формализация данных	11
1.3 Способы хранения данных	14
1.4 Ролевая модель	17
1.5 Системы управления базами данных	18
1.5.1 SurrealDB	18
1.5.2 Neo4j	19
1.6 ArangoDB	19
1.7 Выбор СУБД для решения задачи	20
2 Конструкторский раздел	22
2.1 Проектирование базы данных для хранения Тестов Тьюринга .	22
2.2 Структура разрабатываемого ПО	26
3 Технологический раздел	28
3.1 Средства реализации	28
3.2 Детали реализации	29
3.3 Интерфейс программы	40
4 Исследовательский раздел	44
4.1 Постановка задачи исследования	44

4.1.1	Цель исследования	44
4.1.2	Описание исследования	44
4.1.3	Технические характеристики	46
4.1.4	Результаты исследования	47
ЗАКЛЮЧЕНИЕ		53
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		54
ПРИЛОЖЕНИЕ А		59

ОПРЕДЕЛЕНИЯ

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

Natural Language Processing — «Обработка текстов на естественном языке» относится к области компьютерных наук, а точнее к области искусственного интеллекта или ИИ, связанной с предоставлением компьютерам возможность понимать текст и произносимые слова почти так же, как люди. НЛП сочетает в себе вычислительную лингвистику — моделирование человеческого языка на основе правил — со статистическими моделями, машинным обучением и моделями глубокого обучения. Вместе эти технологии позволяют компьютерам обрабатывать человеческий язык в виде текстовых или голосовых данных и «понимать» его полное значение, включая намерения и чувства говорящего или пишущего. [1]

ACID — в контексте обработки транзакций аббревиатура ACID относится к четырем ключевым свойствам транзакции: атомарность (Atomicity), непротиворечивость (Consistency), изоляция (Isolation) и устойчивость (Durability). [2].

NoSQL — подход к проектированию баз данных, который фокусируется на предоставлении механизма хранения и извлечения данных, который моделируется средствами, отличными от табличных отношений, используемых в реляционных базах данных. Вместо типичной табличной структуры реляционной базы данных базы данных, NoSQL содержит данные в рамках одной структуры данных. Поскольку такая конструкция нереляционной базы данных не требует схемы, она обеспечивает быструю масштабируемость для управления большими и обычно неструктурированными наборами данных [3].

NewSQL — класс современных реляционных СУБД, которые стремятся обеспечить ту же масштабируемую производительность, что и NoSQL, для рабочих нагрузок чтения-записи OLTP, сохраняя при этом гарантии ACID для транзакций. Другими словами, эти системы хотят достичь той же масштабируемости, что и СУБД NoSQL 2000х годов, но при этом сохранить реляционную модель (с SQL) и поддержку транзакций устаревших СУБД 1970х – 1980х годов [4].

Universally Unique Identifier — «Универсальный уникальный идентификатор» — метка, используемая для уникальной идентификации ресурса среди всех других ресурсов этого типа. Является 128-битным значением, обычно представленным в виде 36-буквенной строки [5].

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

NLP — «Natural Language Processing».

ACID — «Atomicity, Consistency, Isolation, Durability».

SQL — «Structured Query Language».

GDB — «Graph Database»

ИИ — «Искусственный Интеллект»

UUID — «Universally Unique IDentifier»

ВВЕДЕНИЕ

Фраза «Тест Тьюринга» наиболее правильно используется для обозначения предложения, сделанного Тьюрингом (1950) как способ решения вопроса о том, могут ли машины мыслить [6]. Согласно Тьюрингу, вопрос о том, могут ли машины мыслить, сам по себе «слишком бессмыслен», чтобы заслуживать обсуждения [7]. Для проведения теста Тьюринга используется программное обеспечение, которое имитирует человеческое поведение и должно убедить эксперта в том, что он общается с живым человеком [7]. Результаты проведения тестов могут быть использованы для разработки и улучшения алгоритмов искусственного интеллекта [8].

Примерами таких алгоритмов ИИ являются алгоритмы обработки естественных языков (Natural Language Processing — NLP). NLP — это совокупность методов и техник, которые позволяют компьютерам анализировать, понимать и генерировать естественный язык [1]. NLP используется в ряде приложений, включая автоматический перевод, распознавание речи и анализ текста [1]. Анализ результатов тестирования поможет в будущем улучшить данные модели, позволяя избегать различные грамматические, орфографические и смысловые ошибки.

Модели от команды OpenAI и многие другие играют важную роль в развитии искусственного интеллекта [9]. GPT-3 используется для создания художественной литературы, поэзии, пресс-релизов, кода, музыки, шуток, технических руководств и новостных статей [10], как предполагает Чалмерс [11], GPT-3 «предлагает потенциальный бездумный путь к общему искусственному интеллекту». Но, конечно, GPT-3 даже не близок к прохождению теста Тьюринга: на глобальном уровне — учитывая значения нескольких предложений, абзацев или двустороннего диалога — становится очевидным, что «GPT-3 пишет текст, в продолжение введенных слов, без какого-либо понимания» [12]. У него нет здравого смысла [13] или способности отслеживать

объекты во время обсуждения [14].

Цель данной работы — разработать базу данных для хранения результатов проведения тестов Тьюринга. База данных должна содержать информацию о тестируемых программах, экспертах, результатах проведения тестов и другие данные, необходимые для анализа результатов.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- Проанализировать предметную область.
- Определить структуру базы данных и ее таблиц.
- Выбрать модели данных и СУБД для реализации разрабатываемой базы данных.
- Написать запросы для вставки, обновления и удаления данных в таблицах.
- Реализовать функционал для получения результатов проведенных тестов, с возможностью фильтрации и сортировки по различным полям.
- Разработать интерфейс пользователя для использования базы данных.
- Провести исследование эффективности запросов на основе разработанной базы данных.

1 Аналитический раздел

В данном разделе описана структура теста Тьюринга. Представлен анализ способов хранения данных и систем управления базами данных.

1.1 Анализ предметной области

Тест Тьюринга — это простой метод определения того, может ли машина продемонстрировать человеческий интеллект [15]. На рисунке 1 представлено схематическое изображение этого метода.

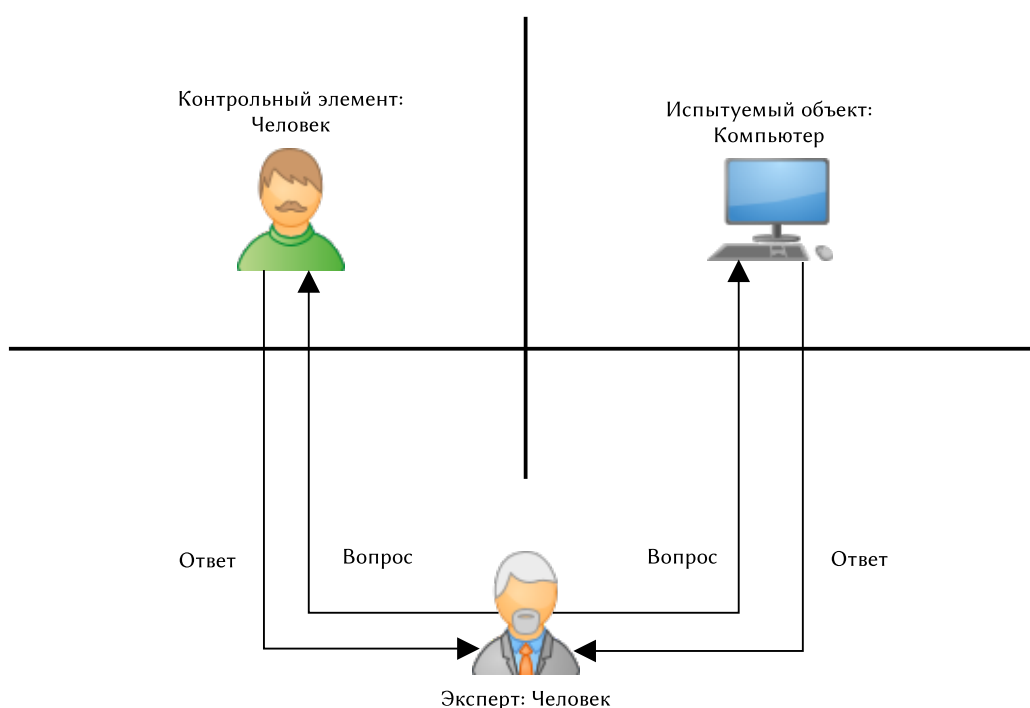


Рисунок 1 — Тест Тьюринга

Тьюринг в своей работе [7] описывает следующий вид игры. Пусть есть человек, машина и эксперт. Эксперт находится в комнате, отделенной от другого человека и машины. Цель игры состоит в том, чтобы эксперт определил, кто из двух является человеком, а кто машиной. Эксперт знает человека и машину по меткам «X» и «Y» — но, по крайней мере в начале игры, не знает, кто из них человек и кто — машина — и в конце игры он должен сказать либо «X — это человек, а Y — машина», либо «X — это машина, а Y — человек».

Эксперту разрешается задавать человеку и машине вопросы следующего вида: «Скажите, пожалуйста, X, играет ли X в шахматы?» Кто бы из машины и другого человека ни был X, он должен отвечать на вопросы, адресованные X. Цель машины состоит в том, чтобы попытаться заставить эксперта ошибочно заключить, что машина — это другой человек; цель другого человека состоит в том, чтобы попытаться помочь эксперту правильно идентифицировать машину [6].

1.2 Формализация данных

Для решения задачи хранения теста Тьюринга необходимо хранить следующие данные:

1. Данные о человеке, машине и эксперте;

1.1 Для человека:

- Имя;
- Пол;
- Возраст;
- Национальность.

1.2 Для машины:

- Модель Искусственного Интеллекта;
- Разработчик модели.

1.3 Для эксперта:

- Имя;
- Пол;
- Возраст;
- Национальность.

2. Данные о заданных вопросах и полученных ответах, а также вердикт эксперта с предположением о сущностях ответчиков;

- Для вопросов — текст вопроса.
- Для ответов — текст ответа.
- Для вердикта — корректность вердикта.

3. Данные об отдельном эксперименте:

- Длительность проведения эксперимента;
- Время, отведенное на дачу ответов субъектами.

4. Данные, о связях между сущностями:

- Последовательность вопросов - ответов, оканчивающаяся вердиктом;
- Связь между ответом и теми, кто его дал;
- Связь между вопросом и теми, кто его задал;
- Связь между вердиктом, тем кто его задал, и относительно каких субъектов он был вынесен;
- Связь между экспериментом и всеми сущностями включенными в данный эксперимент.

На рис. 2 изображена ER-диаграмма предметной области.

1.3 Способы хранения данных

Поскольку в конце теста выносится вердикт о том, является ли отвечающий машиной или человеком, необходимо также хранить, какие ответы были даны, в каком порядке и каким актором. Эти данные должны быть доступны для обработки и сравнения в процессе игры.

Один из способов хранения данных — это использование реляционных баз данных. В этом случае можно создать таблицы для каждого объекта (люди, машины и эксперты, вопросы и ответы, и т.д.) и связать их отношениями. Например, таблицы «Person», «Computer» и «Interrogator» могут быть связаны через внешние ключи. Это является надежным и проверенным способом хранения данных.

Однако более эффективный способ хранения сильносвязанных данных — это использование графовых баз данных (GDB), таких как **SurrealDB** или **Neo4j** [16]. В этом случае каждый объект может быть представлен узлом графа, а отношения между объектами — ребрами графа. Схематическое изображение графовой модели данных можно увидеть на рисунке 3.

Ключевым понятием такой базы данных является граф (или ребро, или взаимосвязь). Граф связывает элементы данных в хранилище с набором узлов и ребер, причем ребра представляют отношения между узлами. Отношения позволяют напрямую связывать данные в хранилище и во многих случаях извлекать их с помощью одной операции. Графовые базы данных удерживают отношения между данными в качестве приоритета. Запросы по отношениям проходят быстро, потому что они постоянно хранятся в базе данных в качестве отдельных сущностей. Отношения можно визуализировать с помощью графов, что делает их полезными для сильно взаимосвязанных данных [17]. В графовых базах данных нет необходимости использовать сложные JOIN-запросы, что может существенно упростить запросы к данным по отношениям.

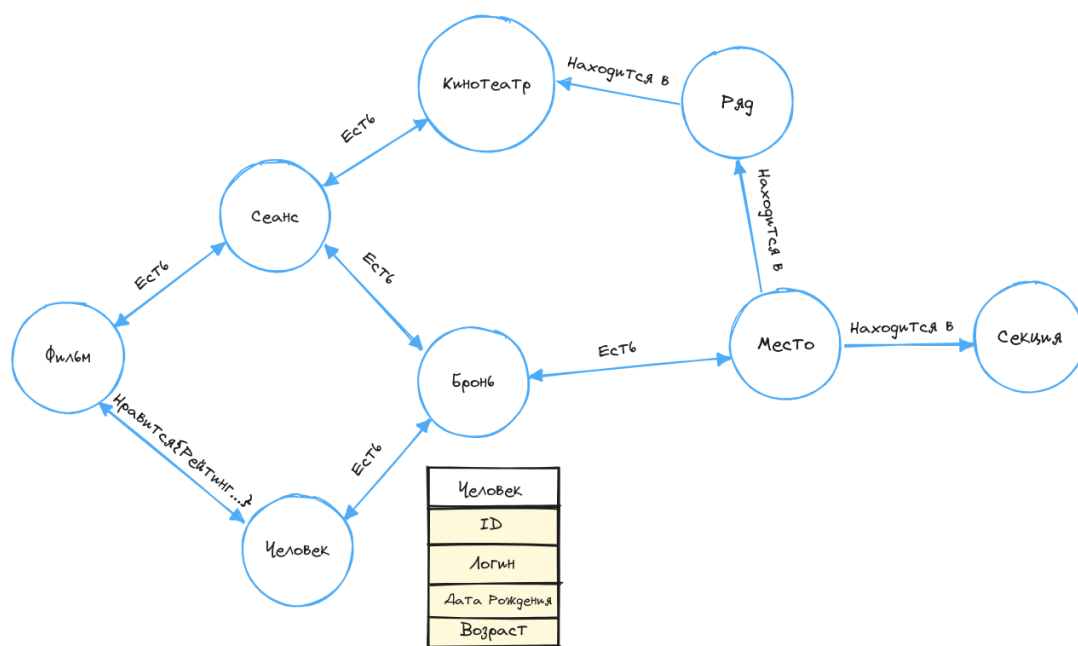


Рисунок 3 – Представление базы данных в виде графа

Графовые базы данных также обеспечивают быстрый доступ к данным по отношениям, что делает их эффективными при работе с глубоко связанными данными. Они также позволяют легко добавлять новые данные в граф без необходимости изменения схемы базы данных. Однако, реляционные базы данных обладают более высокой надежностью и могут обеспечивать лучшую производительность при выполнении сложных запросов, особенно если используются правильно настроенные индексы.

Таким образом, выбор между реляционными базами данных и графовыми зависит от конкретных требований проекта. В контексте данной работы, графовая модель подходит больше, чем реляционная, потому что тест Тьюринга включает в себя множество связей между объектами (человек, машина, эксперт, вопросы и ответы и т.д.), которые можно представить в виде графа.

Графовая модель также становится еще более привлекательной, если вспомнить какая идея была обозначена в начале данной работы: создание

инструмента для улучшения искусственного интеллекта. Результаты тестов Тьюринга, которые будут храниться в графовой базе данных, можно будет использовать для эффективного, по сравнению с традиционными реляционными базами данных, обучения ИИ. Эффективность достигается за счет того, что данные, используемые для обучения ИИ, представлены в виде графа, а не в виде стандартных «табличных» значений [18]. Таким образом, схожая структура данных внутри БД поможет разработать более гибкую и быстродействующую систему при меньших затраченных ресурсах.

На рис. 4 можно увидеть результаты сравнения 3 различных баз данных: реляционной (**PostgreSQL**), графовой (**Neo4j**) и мультимодельной (**ArangoDB**). Исходя из графика видно, что графовая, а также мультимодельная база данных обеспечивает лучшую производительность при выполнении запросов по отношениям (**JOIN-запросов**), а также **PROJECTION-запросов**. Скорость графовой и мультимодельной баз данных относительно реляционной по **PROJECTION-запросам** объясняется особенностью **key-value** хранилищ, лежащих в основе графовой и мультимодельной баз данных. При этом, реляционная база данных обеспечивает лучшую производительность при выполнении операций, не связанных с отношениями, а именно **ORDER BY** и **AGGREGATION**.

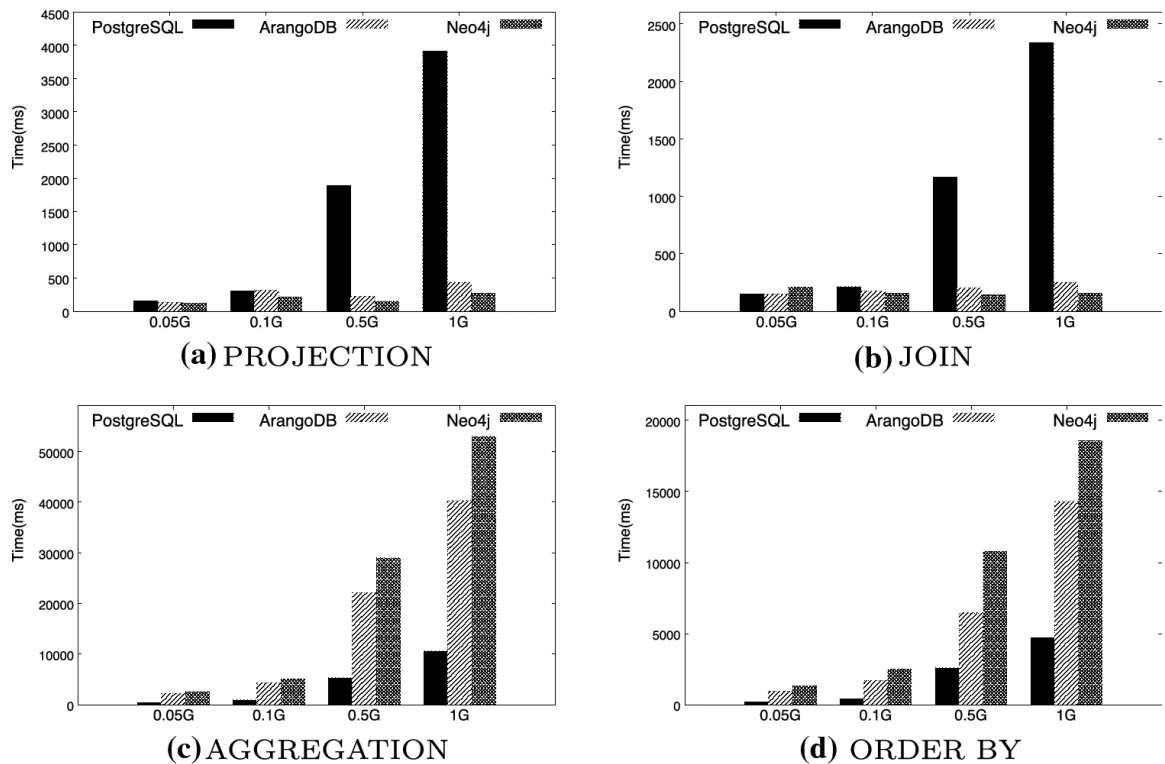


Рисунок 4 – Сравнение времени выполнения атомарных операций над различными базами данных.

1.4 Ролевая модель

В системе определены четыре возможные роли, ограничивающие доступ к получению/добавлению информации.

1. Эксперт — пользователь, обладающий возможностью задавать новые вопросы подопытным, а также выносить вердикт на основе полученных решений, имеет доступ к прочтению всех данных в рамках сессии;
2. Компьютер — пользователь, обладающий возможностью дачи ответов на вопросы эксперта, имеет доступ к чтению своих прошлых ответов и вопросов к ним в рамках сессии;
3. Человек — пользователь, обладающий равными с компьютером правами и возможностями. Ввиду различия характеристик между человеком и компьютером, в системе необходимо отдельно хранить данные об этих сущностях, а также выделить отдельные роли;
4. Администратор — пользователь, обладающий возможностью добавле-

ния/удаления/изменения всех пользователей, сущностей и полей базы данных, а также просмотра всех данных в рамках базы данных.

На рисунке 5 представлена диаграмма ролей.

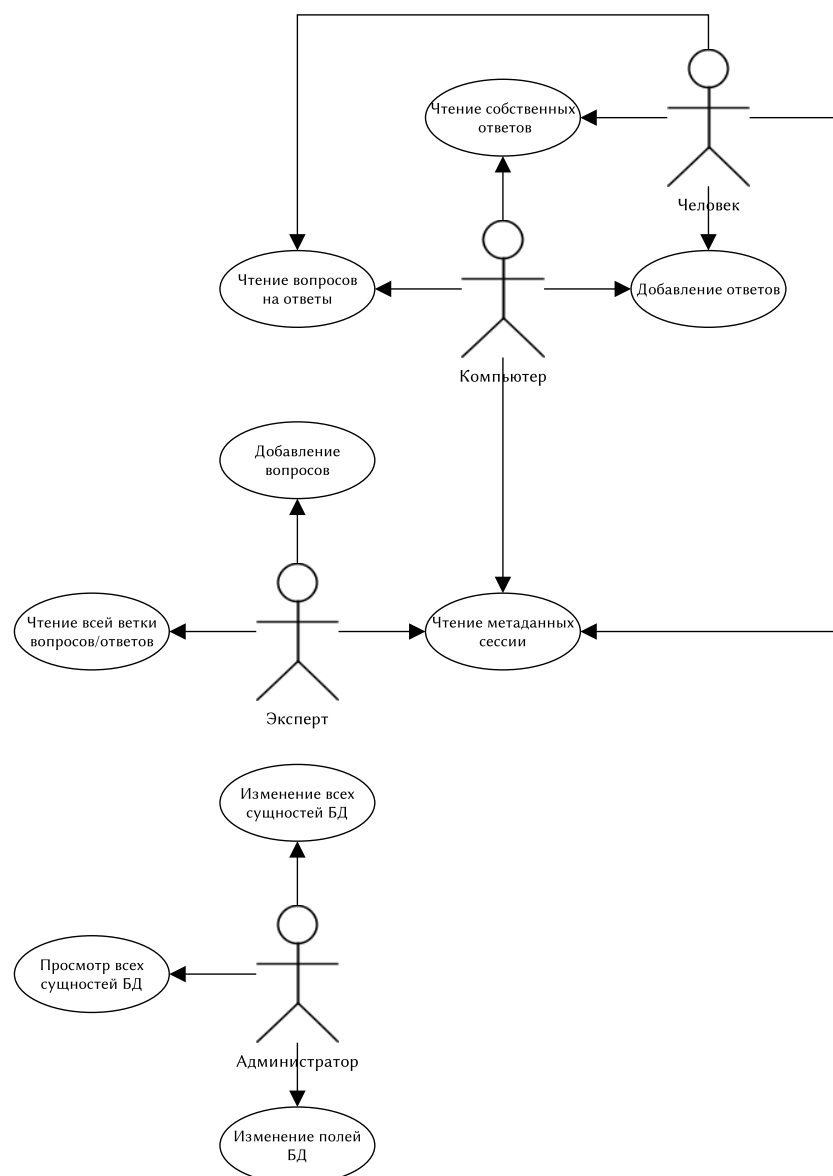


Рисунок 5 – Диаграмма ролей

1.5 Системы управления базами данных

1.5.1 SurrealDB

SurrealDB — мультимodelьная NewSQL база данных, которая работает в режиме полной схемы (SCHEMAFULL) или без схемы (SCHEMALESS), с таблицами, ссылками на записи между документами (без JOIN) и функциями моделирования базы данных на основе графов [19].

Благодаря использованию **SurrealDB** особых методов сегментирования и репликации, становится возможным повысить производительность за счет распределения нагрузки между несколькими компьютерами [20].

Также особая архитектура базы данных позволяет работать как в оперативной памяти (**in-memory**), на дисковом пространстве (**on-disk**) или как распределенная база данных, используя **TiKV** [21].

Поскольку **SurrealDB** — мультимодельная база данных, становятся также возможными классические реляционные методики проектирования баз данных, что повышает гибкость итоговой системы.

1.5.2 Neo4j

Neo4j — это графовая база данных, которая позволяет хранить, управлять и анализировать связанные данные. Она была разработана с учетом графовой модели данных, в которой данные представлены в виде узлов (вершин) и связей (ребер) [22].

Одним из преимуществ **Neo4j** является то, что она позволяет эффективно моделировать и анализировать сложные связи между данными, такие как в социальных сетях, географических картах и сетях предприятий. Это делает ее очень полезной для приложений, которые требуют быстрого доступа к сложным связным данным и быстрой обработки запросов.

Однако поскольку **Neo4j** — исключительно графовая база данных, хранение и получение данных без каких-либо связей друг с другом может вызывать проблемы с производительностью, вне зависимости от размера запроса [23].

1.6 ArangoDB

Подобно **SurrealDB**, **ArangoDB** также является мультимодельной базой данных, однако при этом является **NoSQL** базой данных в противовес **NewSQL**.

ArangoDB — был разработан специально для обслуживания различных типов баз данных **NoSQL**. Таким образом, варианты использования **ArangoDB** могут включать одновременную разработку баз данных, ориентированных на

ключ, граф или документ [24].

Отличительной чертой всех NoSQL баз данных является отсутствие возможности объявления схемы данных внутри таблицы, и ArangoDB не исключение. Несмотря на то, что у ArangoDB есть внешний инструмент валидации данных через JSON-схемы, данная возможность проигрывает в гибкости настройки полноценным схемам традиционных реляционных и NewSQL баз данных. Для непосредственного сравнения, документация SurrealDB [25] насчитывает порядка 30 базовых типов, спецификация JSON-схемы [26] насчитывает всего лишь 7.

1.7 Выбор СУБД для решения задачи

Среди графовых баз данных можно выделить три наиболее подходящие системы: SurrealDB, Neo4j и ArangoDB. Эти три СУБД обеспечивают быстрый доступ к данным по отношениям, что делает их эффективными при работе с глубоко связанными данными.

Однако SurrealDB имеет дополнительные преимущества перед Neo4j и ArangoDB. В противовес Neo4j, она является мультимodelьной базой данных, что позволяет эффективное хранение и получение несвязанных данных, например выгрузка всех экспертов, которые участвовали в различных экспериментах. В таких случаях, Neo4j может испытывать определенные проблемы связанные в первую очередь с производительностью.

Если сравнивать SurrealDB с ArangoDB, то SurrealDB предоставляет более гибкий инструмент настройки схемы базы данных.

Вывод

В данном разделе:

- рассмотрена сущность и структура теста Тьюринга;
- определены ключевые роли в рамках эксперимента, связанного с тестом Тьюринга;
- проанализированы способы хранения информации для системы и выбраны оптимальные способы для решения поставленной задачи;

- были рассмотрены два различных типа баз данных: реляционные и графовые;
- было выявлено, что для решения задачи теста Тьюринга наиболее подходящей является графовая база данных, а конкретно **SurrealDB**.

2 Конструкторский раздел

В данном разделе представлены этапы проектирования выделенной в предыдущем разделе базы данных, нужной для решения задачи.

2.1 Проектирование базы данных для хранения Тестов Тьюринга

База данных для хранения Тестов Тьюринга будет реализована с использованием СУБД SurrealDB. В базе данных будет существовать 7 таблиц и 7 типов отношений. Схема разработанной базы данных представлена на рисунке 6.

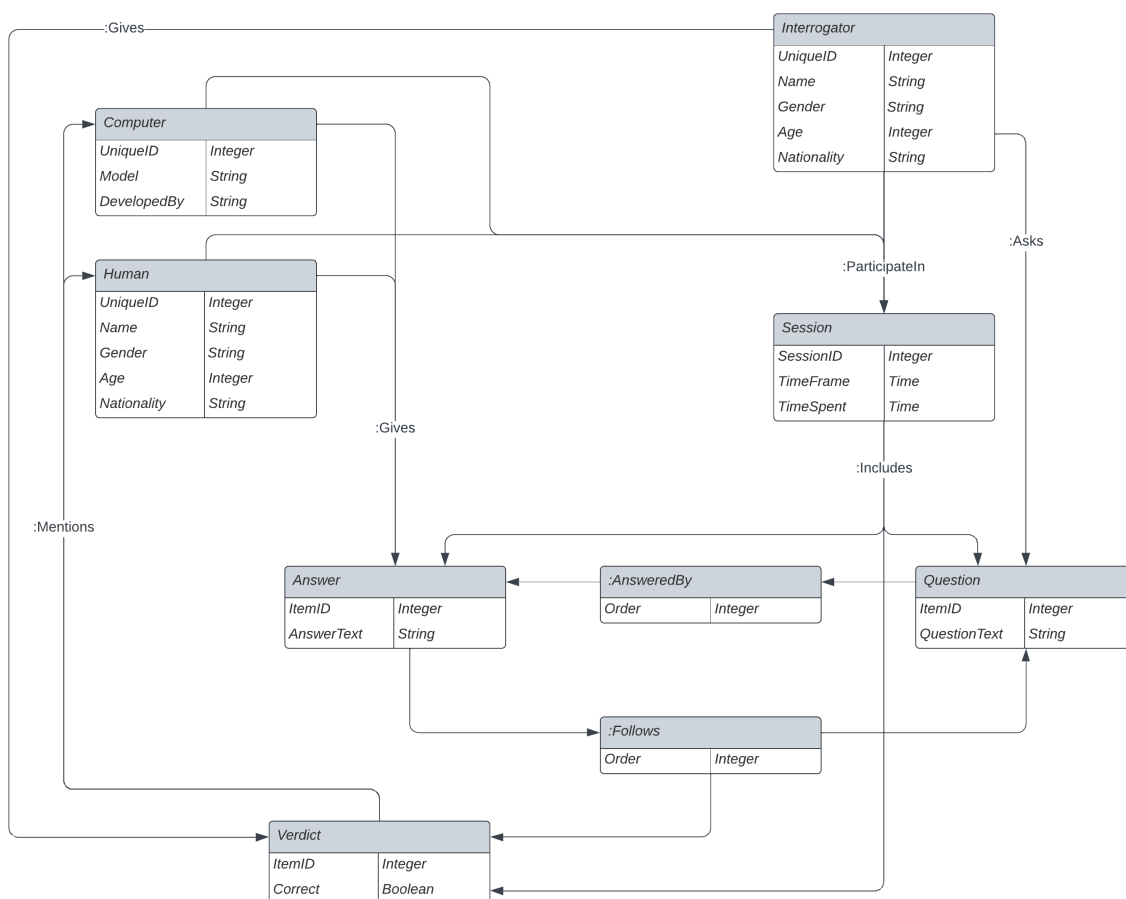


Рисунок 6 – Схема разработанной базы данных.

Поля таблицы **Interrogator**:

1. **UniqueID** — уникальный идентификатор — UUID;
2. **Name** — имя «эксперта» — строка;

3. **Gender** — пол «эксперта» — строка;
4. **Age** — возраст «эксперта» — целое число;
5. **Nationality** — национальность «эксперта» — строка.

Данная таблица отвечает за хранение данных, связанных с экспертом, проводящим эксперимент. Эта таблица связана со следующими таблицами:

- **Session** — через отношение **:ParticipateIn**;
- **Answer** — через отношение **:Asks**;
- **Verdict** — через отношение **:Gives**.

Поля таблицы **Computer**:

1. **UniqueID** — уникальный идентификатор — UUID;
2. **Model** — Модель ИИ, проходившая тест — строка;
3. **DevelopedBy** — Разработчики указанной модели ИИ — строка.

Данная таблица отвечает за хранение данных, связанных с компьютером, участвующем в эксперименте. Эта таблица связана со следующими таблицами:

- **Session** — через отношение **:ParticipateIn**;
- **Answer** — через отношение **:Gives**;
- **Verdict**. — через отношение **:Mentions**.

Поля таблицы **Human**:

1. **UniqueID** — уникальный идентификатор — UUID;
2. **Name** — имя человека — строка;
3. **Gender** — пол человека — строка;
4. **Age** — возраст человека — строка;
5. **Nationality** — национальность человека — строка.

Данная таблица отвечает за хранение данных, связанных с человеком, участвующем в эксперименте. Эта таблица связана со следующими таблицами:

- **Session** — через отношение **:ParticipateIn**;
- **Answer** — через отношение **:Gives**;

- **Verdict** — через отношение **:Mentions**.

Поля таблицы **Answer**:

1. **ItemID** — уникальный идентификатор — **UUID**.
2. **AnswerText** — текст ответа — строка.

Данная таблица отвечает за хранение данных, связанных с ответами, данными в эксперименте. Следует отметить, что ответы, данные на протяжении всех экспериментов, являются уникальными сущностями, или, иными словами, в базе данных нет двух одинаковых ответов на любой из вопросов. Данная особенность преследует цель показать связь между вопросами и ответами, и как различные вопросы могут привести к одним и тем же ответам, либо же, как компьютер и человек в эксперименте могут дать одинаковый ответ.

Эта таблица связана со следующими таблицами:

- **Session** — через отношение **:Includes**;
- **Answer** — через отношения **:AnsweredBy** и **:Follows**;
- **Computer** — через отношение **:Gives**;
- **Human** — через отношение **:Gives**;
- **Verdict** — через отношение **:Follows**.

Поля таблицы **Question**:

1. **ItemID** — уникальный идентификатор — **UUID**.
2. **QuestionText** — текст вопроса — строка.

Данная таблица отвечает за хранение данных, связанных с вопросами, данными в эксперименте экспертом. Следует отметить, что вопросы, данные на протяжении всех экспериментов, как и ответы, упомянутые выше, являются уникальными сущностями, или, иными словами, в базе данных нет двух одинаковых вопросов. Данная особенность преследует цель показать связь между вопросами и ответами, и как на один вопрос можно привести множество различных ответов.

Эта таблица связана со следующими таблицами:

- **Session** — через отношение **:Includes**;
- **Answer** — через отношения **:AnsweredBy** и **:Follows**;
- **Interrogator** — через отношение **:Asks**.

Поля таблицы **Verdict**:

1. **ItemID** — уникальный идентификатор — **UUID**;
2. **Correct** — Верен ли вердикт, выданный экспертом — ложь / истина.

Данная таблица отвечает за хранение данных, связанных с вердиктами, данными экспертами по окончанию экспериментов. После любого данного ответа, эксперт может закончить эксперимент и выдать свой вердикт, кто является компьютером, а кто человеком.

Эта таблица связана со следующими таблицами:

- **Session** — через отношение **:Includes**;
- **Answer** — через отношение **:Follows**;
- **Interrogator** — через отношение **:Gives**;
- **Computer** — через отношение **:Mentions**;
- **Human** — через отношение **:Mentions**.

Поля таблицы **Session**:

1. **SessionID** — уникальный идентификатор — **UUID**;
2. **TimeFrame** — период времени, отведенный на ответ на вопрос — время;
3. **TimeSpent** — продолжительность сессии — время.

Данная таблица отвечает за хранение данных, связанных с различными экспериментами. Данная таблица является своего рода мета-таблицей, по связи с которой можно получить данные о всех сущностях, участвующих в эксперименте.

Эта таблица связана со следующими таблицами:

- **Question** — через отношение **:Includes**;
- **Answer** — через отношение **:Includes**;
- **Answer** — через отношение **:Includes**;
- **Interrogator** — через отношение **:ParticipateIn**;

- **Computer** — через отношение `:ParticipateIn`;
- **Human** — через отношение `:ParticipateIn`.

Поля рёбер `:AnsweredBy` и `:Follows`:

1. **Order** — порядковый номер вопроса/ответа/вердикта в системе - целое число

Особенность **SurrealDB** заключается в том, что отношения также могут иметь дополнительные поля, характеризующие их. Поле **Order** необходимо для построения контекста ответов/вопросов поскольку ответ может различаться от того, какие ответы были даны ранее.

2.2 Структура разрабатываемого ПО

Предполагается, что разрабатываемый проект является одним цельным **Electron**-подобным [27] приложением. Серверная часть и графический интерфейс упаковывается в единый бинарный файл, предоставляя возможность создать различные версии приложения под различные операционные системы. В общем смысле, серверная часть коммуницирует с базой данных, доставляя результат к графическому интерфейсу в рамках единого приложения.

Общая схема архитектура приложения представлена на рисунке 7

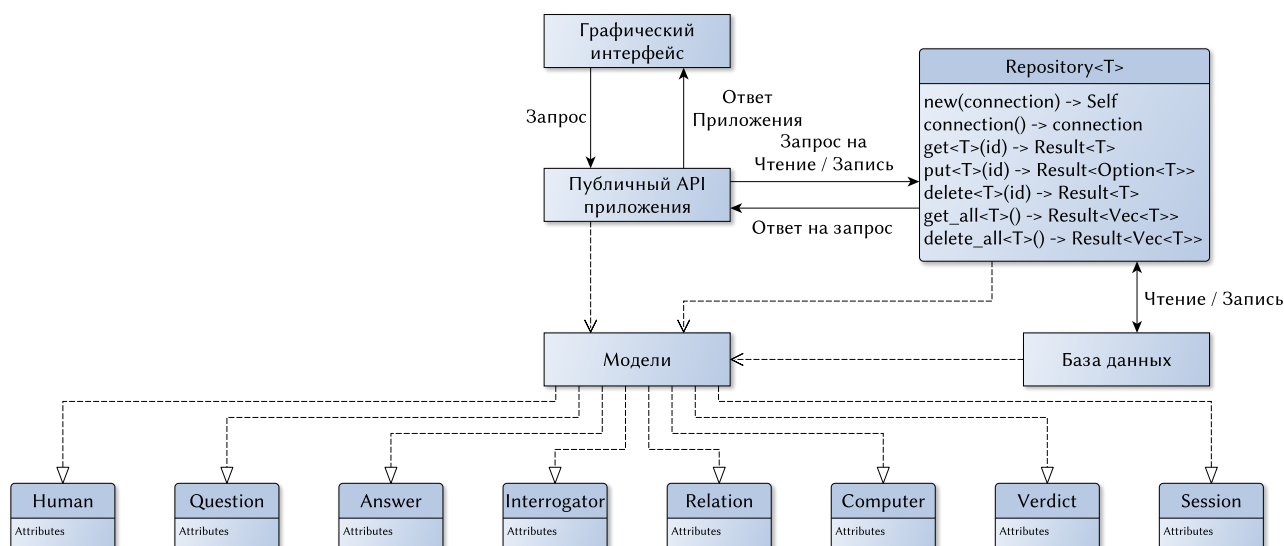


Рисунок 7 – Схема архитектуры приложения

Вывод

В данном разделе были представлены проектирование базы данных, рассмотрены особенности используемой СУБД на архитектурном уровне и была показана структура разрабатываемого ПО.

3 Технологический раздел

3.1 Средства реализации

Основным языком программирования является мультипарадигменный язык Rust [28].

- Одно из главных достоинств данного языка это гарантия безопасной работы с памятью при помощи системы статической проверки ссылок, так называемый **Borrow Checker** [29].
- Отсутствие сборщика мусора, как следствие, более экономная работа с ресурсами.
- Встроенный компилятор, поставляемый совместно с пакетным менеджером **Cargo**.
- Кросс-платформенность, от **UNIX** и **MacOS** приложений до **Web** - приложений.
- **SurrealDB** написан на языке **Rust**, в следствии чего инструментарий языка наиболее плотно работает с непосредственно самой базой данных.
- Важно отметить, что язык программирования **Rust** сопоставим по скорости с такими языками как **C** [30] и **C++** [31], предоставляя в то же время более широкий функционал для тестирования кода и контроля памяти.

Также в рамках языка **Rust** был выбран фреймворк **Tauri**. **Tauri** используется для создания приложений с использованием комбинации инструментов **Rust** и **HTML**, отображаемых в **Webview**. Приложения, созданные с помощью **Tauri**, могут поставляться с любым количеством дополнительных **JS API** и **Rust API**, так что **Webview** может управлять системой посредством передачи сообщений. Разработчики могут расширить **API** за счет своей собственной функциональности и легко объединить **Webview** и серверную часть на основе **Rust** [32]. На рисунке 8 изображена общая архитектура **Tauri**-приложения.

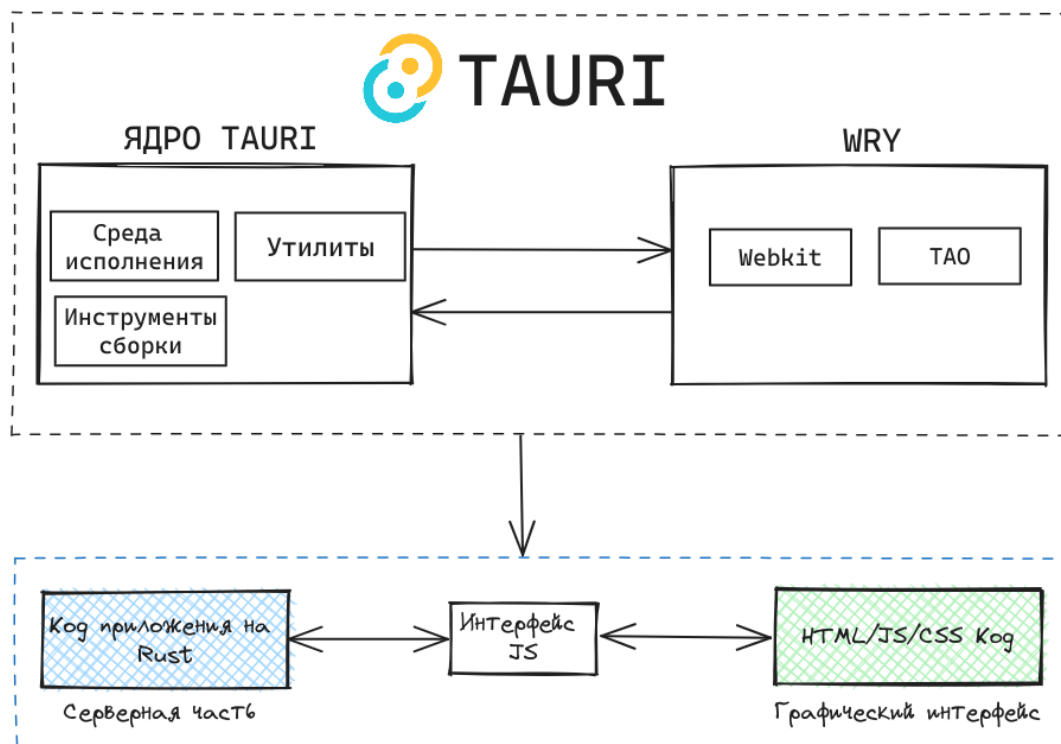


Рисунок 8 – Принцип работы Tauri

3.2 Детали реализации

На листинге 1 представлены трейты, аналог интерфейса в языке Rust, необходимые для реализации репозитория, который отвечает за взаимодействие между публичным API и непосредственно с базой данных.

Листинг 1 – Трейты, необходимые для реализации репозитория.

```
use crate::models::{HasRole, User};
use crate::prelude::*;
use ::surrealdb::opt::auth::Jwt;
use std::error::Error;
use uuid::Uuid;

pub mod surrealdb;

pub trait CrudOps<T> {
    async fn get(&self, id: Uuid) -> Result<T, Box<dyn Error>>;
    async fn put(&self, id: Uuid, value: T) -> Result<Option<T>, Box<dyn Error>>;
    async fn delete(&self, id: Uuid) -> Result<T, Box<dyn Error>>;
    async fn get_all(&self) -> Result<Vec<T>, Box<dyn Error>>;
    async fn delete_all(&self) -> Result<Vec<T>, Box<dyn Error>>;
}
```

```

}

pub trait MetaOps {
    async fn get_meta(&self) -> Result<User, Box<dyn Error>>;
}

pub trait Repository<T, C>
where
    C: CrudOps<T>,
{
    fn new(connection: C) -> Result<Self>
    where
        Self: std::marker::Sized;
    fn connection(&self) -> C;
    async fn get<G>(&self, id: Uuid) -> Result<T, Box<dyn Error>> {
        self.connection().get(id).await
    }
    async fn put(&self, id: Uuid, value: T) -> Result<Option<T>, Box<dyn Error>> {
        self.connection().put(id, value).await
    }

    async fn delete(&self, id: Uuid) -> Result<T, Box<dyn Error>> {
        self.connection().delete(id).await
    }

    async fn get_all(&self) -> Result<Vec<T>, Box<dyn Error>> {
        self.connection().get_all().await
    }

    async fn delete_all(&self) -> Result<Vec<T>, Box<dyn Error>> {
        self.connection().delete_all().await
    }
}

pub trait Utils<T, C>
where
    Self: Repository<T, C>,
    C: MetaOps + CrudOps<T>,
    T: HasRole,
{

```

```

    async fn meta_from_jwt(&self, token: Jwt) -> Result<User, Box<dyn Error>> {
        self.connection().get_meta().await
    }
}

```

На листинге 2 представлена реализация репозитория и моделей, необходимых для трансляции данных из SurrealDB в пространство языка Rust.

Листинг 2 – Реализация репозитория.

```

impl<T: DeserializeOwned + Serialize + Send + Sync + HasId> CrudOps<T> for Surreal<Client> {
    async fn get(&self, id: Uuid) -> Result<T, Box<dyn Error>> {
        let result: Option<T> = self
            .select((
                std::any::type_name::<T>()
                    .to_lowercase()
                    .rsplit("::")
                    .next()
                    .unwrap_or(&std::any::type_name::<T>().to_lowercase()),
                id.to_string(),
            ))
            .await?;

        result.map_or_else(
            || {
                tracing::warn!("GET: ␣field␣not␣found");
                Err(Err::GetNotFound {
                    table: std::any::type_name::<T>()
                        .to_lowercase()
                        .rsplit("::")
                        .next()
                        .unwrap_or(&std::any::type_name::<T>().to_lowercase())
                        .to_string(),
                    id,
                })
                .into()
            },
            |mut res| {
                tracing::info!("GET: ␣success");
                *res.id() = id;
                Ok(res)
            }
        )
    }
}

```

```

    },
)
}

async fn put(&self, id: Uuid, value: T) -> Result<Option<T>, Box<dyn Error>> {
    let result: Option<T> = self
        .update((
            std::any::type_name::<T>()
                .to_lowercase()
                .rsplit("::")
                .next()
                .unwrap_or(&std::any::type_name::<T>().to_lowercase()),
            id.to_string(),
        ))
        .content(value)
        .await?;

    Ok(result)
}

async fn delete(&self, id: Uuid) -> Result<T, Box<dyn Error>> {
    let result: Option<T> = self
        .delete((
            std::any::type_name::<T>()
                .to_lowercase()
                .rsplit("::")
                .next()
                .unwrap_or(&std::any::type_name::<T>().to_lowercase()),
            id.to_string(),
        ))
        .await?;

    result.map_or_else(
        || {
            tracing::warn!("DELETE: field not found");
            Err(Err::GetNotFound {
                table: std::any::type_name::<T>()
                    .to_lowercase()
                    .rsplit("::")
                    .next()

```



```

        .unwrap_or(&std::any::type_name::<T>().to_lowercase())
        .to_string(),
        id,
    }
    .into())
},
|res| {
    tracing::info!("DELETE:␣success");
    Ok(res)
},
)
}

async fn get_all(&self) -> Result<Vec<T>, Box<dyn Error>> {
    Ok(self
        .select(
            std::any::type_name::<T>()
                .to_lowercase()
                .rsplit("::")
                .next()
                .unwrap_or(&std::any::type_name::<T>().to_lowercase()),
        )
        .await?)
}

async fn delete_all(&self) -> Result<Vec<T>, Box<dyn Error>> {
    Ok(self
        .delete(
            std::any::type_name::<T>()
                .to_lowercase()
                .rsplit("::")
                .next()
                .unwrap_or(&std::any::type_name::<T>().to_lowercase()),
        )
        .await?)
}
}

// impl<T: DeserializeOwned> MetaOps<T> for Surreal<Client> {
impl MetaOps for Surreal<Client> {

```

```

    async fn get_meta(&self) -> Result<User, Box<dyn Error>> {
        let res: SurrealUser = self
            .select(("user"))
            .await?
            .pop()
            .ok_or(Box::from("No meta found") as Box<dyn Error>)?;

        res.try_into()
    }
}

default impl<T: DeserializeOwned + Serialize + Send + Sync + HasId> Repository<T,
    Surreal<Client>>
    for SurrealRepo<T>
{
    fn new(connection: Surreal<Client>) -> Result<Self> {
        Ok(Self {
            connection,
            object: PhantomData:::<T>,
        })
    }

    fn connection(&self) -> Surreal<Client> {
        self.connection.clone()
    }
}

// Needed for specialization
impl Repository<Human, Surreal<Client>> for SurrealRepo<Human> {}
impl Repository<Interrogator, Surreal<Client>> for SurrealRepo<Interrogator> {}
impl Repository<Computer, Surreal<Client>> for SurrealRepo<Computer> {}
impl Repository<Answer, Surreal<Client>> for SurrealRepo<Answer> {}
impl Repository<Question, Surreal<Client>> for SurrealRepo<Question> {}
impl Repository<Session, Surreal<Client>> for SurrealRepo<Session> {}
impl Repository<Verdict, Surreal<Client>> for SurrealRepo<Verdict> {}

```

На листинге 3 представлены запросы, которые вызываются при инициализации базы данных.

Листинг 3 – Инициализация базы данных.

```

REMOVE DATABASE TuringDB;
REMOVE NAMESPACE TuringApp;

DEFINE NAMESPACE TuringApp;
USE NS TuringApp;
DEFINE DATABASE TuringDB;
USE DB TuringDB;

DEFINE TABLE role SCHEMAFULL
  PERMISSIONS
    FOR create, update NONE,
    FOR select WHERE $auth.roles containsany [role:human, role:interrogator,
      role:computer],
    FOR delete NONE;
create role:human;
create role:computer;
create role:interrogator;

DEFINE TABLE user SCHEMAFULL
  PERMISSIONS
    FOR select, update WHERE id = $auth.id,
    FOR create, delete NONE;
DEFINE FIELD user ON user TYPE string;
DEFINE FIELD password ON user TYPE string;
DEFINE FIELD role ON user TYPE record;
DEFINE INDEX idx_user ON user COLUMNS user UNIQUE;

DEFINE SCOPE TuringScope
  SESSION 1h
  SIGNUP ( CREATE user SET id = rand::uuid(), user = $user, password =
    crypto::argon2::generate($password), role = $role )
  SIGNIN ( SELECT * FROM user WHERE user = $user AND crypto::argon2::compare(password,
    $password) );

DEFINE TABLE human SCHEMAFULL
  PERMISSIONS
    FOR select WHERE true,
    FOR create, delete, update WHERE id = $auth.id AND $auth.role containsany
      [role:human];
DEFINE FIELD name ON human TYPE string;

```

```

DEFINE FIELD age ON human TYPE int;
DEFINE FIELD gender ON human TYPE string;
DEFINE FIELD nationality ON human TYPE string;

DEFINE TABLE interrogator SCHEMAFULL
  PERMISSIONS
    FOR select WHERE true,
    FOR create, delete, update WHERE id = $auth.id AND $auth.role containsany
      [role:interrogator];
  DEFINE FIELD name ON interrogator TYPE string;
  DEFINE FIELD age ON interrogator TYPE int;
  DEFINE FIELD gender ON interrogator TYPE string;
  DEFINE FIELD nationality ON interrogator TYPE string;

DEFINE TABLE verdict SCHEMAFULL
  PERMISSIONS
    FOR select WHERE true,
    FOR create, delete, update WHERE ->gives->id = $auth.id AND $auth.role containsany
      [role:interrogator];
  DEFINE FIELD correct ON verdict TYPE bool;

DEFINE TABLE session SCHEMAFULL
  PERMISSIONS
    FOR select WHERE true,
    FOR create, delete, update WHERE ->participateIn->id = $auth.id AND $auth.role
      containsany [role:interrogator];
  DEFINE FIELD time_start ON session TYPE string DEFAULT time::now();
  DEFINE FIELD time_end ON session TYPE string DEFAULT time::now();
  DEFINE FIELD time_spent ON session TYPE string DEFAULT 0;

DEFINE TABLE question SCHEMAFULL
  PERMISSIONS
    FOR select WHERE true,
    FOR create, delete, update WHERE ->asks->id = $auth.id AND $auth.role containsany
      [role:interrogator];
  DEFINE FIELD text ON question TYPE string;

DEFINE TABLE computer SCHEMAFULL
  PERMISSIONS
    FOR select WHERE true,

```

```

        FOR create, delete, update WHERE id = $auth.id AND $auth.role containsany
            [role:computer];
    DEFINE FIELD model ON computer TYPE string;
    DEFINE FIELD developed_by ON computer TYPE string;

    DEFINE TABLE answer SCHEMAFULL
        PERMISSIONS
            FOR select WHERE true,
            FOR create, delete, update WHERE ->gives->id = $auth.id
                AND $auth.role containsany [role:computer, role:human];
    DEFINE FIELD text ON answer TYPE string;

    DEFINE TABLE mentions;
    DEFINE TABLE asks;
    DEFINE TABLE gives;
    DEFINE TABLE includes;
    DEFINE TABLE participateIn;
    DEFINE TABLE follows;
    DEFINE TABLE answeredBy;

    DEFINE FIELD order ON follows TYPE int;
    DEFINE FIELD answeredBy ON follows TYPE int;

    DEFINE INDEX mentions ON TABLE mentions COLUMNS in, out;
    DEFINE INDEX asks ON TABLE asks COLUMNS in, out;
    DEFINE INDEX gives ON TABLE gives COLUMNS in, out;
    DEFINE INDEX includes ON TABLE includes COLUMNS in, out;
    DEFINE INDEX participateIn ON TABLE participateIn COLUMNS in, out;
    DEFINE INDEX follows ON TABLE follows COLUMNS in, out;
    DEFINE INDEX answeredBy ON TABLE answeredBy COLUMNS in, out;

```

На листинге 4 представлены 3 метода API, которые вызываются со стороны графического интерфейса.

Листинг 4 – Методы API.

```

#[derive(Deserialize)]
pub struct Context {
    pub host: String,
    pub port: u16,
    pub ns: String,

```

```

    pub db: String,
    pub sc: String,
}

#[tauri::command]
pub async fn login(
    Credentials {
        username,
        password,
        host,
        port,
        ns,
        db,
        sc,
    }: Credentials,
) -> Result<Jwt, Err> {
    let connection = Surreal::new::<Ws>(format!("{:}:{:}", host, port)).await?;
    let res = connection
        .signin(Scope {
            namespace: &ns,
            database: &db,
            scope: &sc,
            params: LoginParams {
                user: &username,
                password: &password,
            },
        })
        .await?;

    Ok(res)
}

#[tauri::command]
pub async fn signup(
    Credentials {
        username,
        password,
        host,
        port,
        ns,

```

```

        db,
        sc,
    }: Credentials,
    role: Role,
) -> Result<Jwt, Err> {
    tracing::info!("received_credentials: {ns}, {db}, {username}, {host}, {port}, {role:?}");
    let connection = Surreal::new::<Ws>(format!("{host}:{port}", host, port)).await?;
    connection.use_ns(&ns).use_db(&db).await?;
    let sc = Scope {
        namespace: &ns,
        database: &db,
        scope: &sc,
        params: AuthParams {
            user: &username,
            password: &password,
            role: role.into(),
        },
    };
    Ok(connection.signup(sc).await?)
}

#[tauri::command]
pub async fn get_info(
    Context {
        host,
        port,
        ns,
        db,
        sc,
    }: Context,
    token: Jwt,
) -> Result<User, Err> {
    tracing::info!("received_credentials: {ns}, {db}, {host}, {port}");
    let connection = Surreal::new::<Ws>(format!("{host}:{port}", host, port)).await?;
    connection.use_ns(&ns).use_db(&db).await?;
    connection.authenticate(token).await?;

    connection.get_meta().await.map_err(Err::General)
}

```

3.3 Интерфейс программы

Для работы с БД был разработан графический интерфейс. Для реализации интерфейса был использован фреймворк `Next.js` с использованием языка `Typescript`. В программного интерфейсе реализованы простейшие операции, связанные с просмотром данных об эксперимента. Также добавлены функции входа - выхода из системы. Графический интерфейс представлен на рисунках 9, 10, 12, 13 и 11.

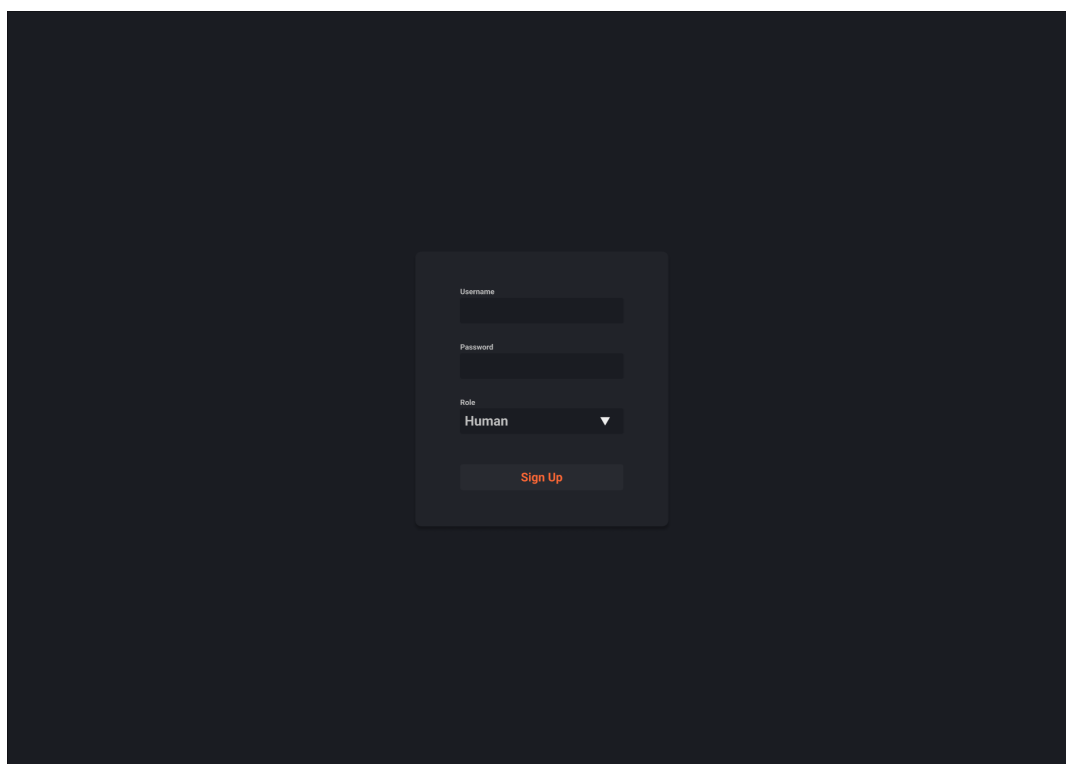


Рисунок 9 – Страница регистрации нового пользователя

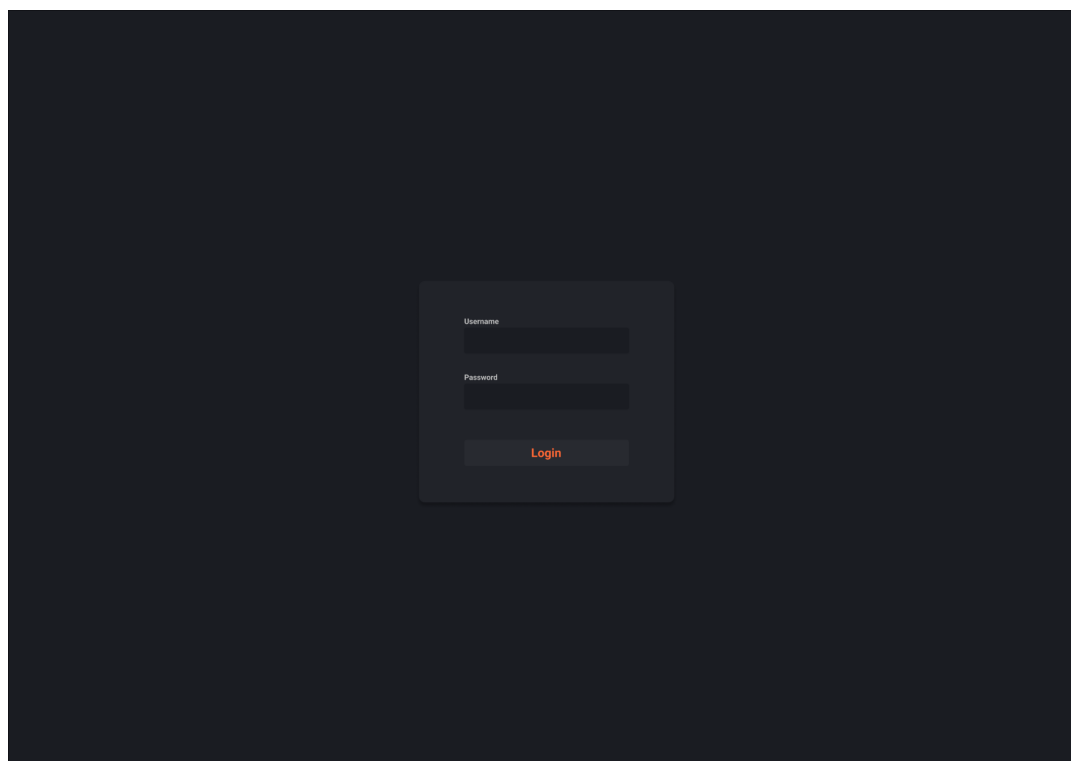


Рисунок 10 – Страница входа в систему

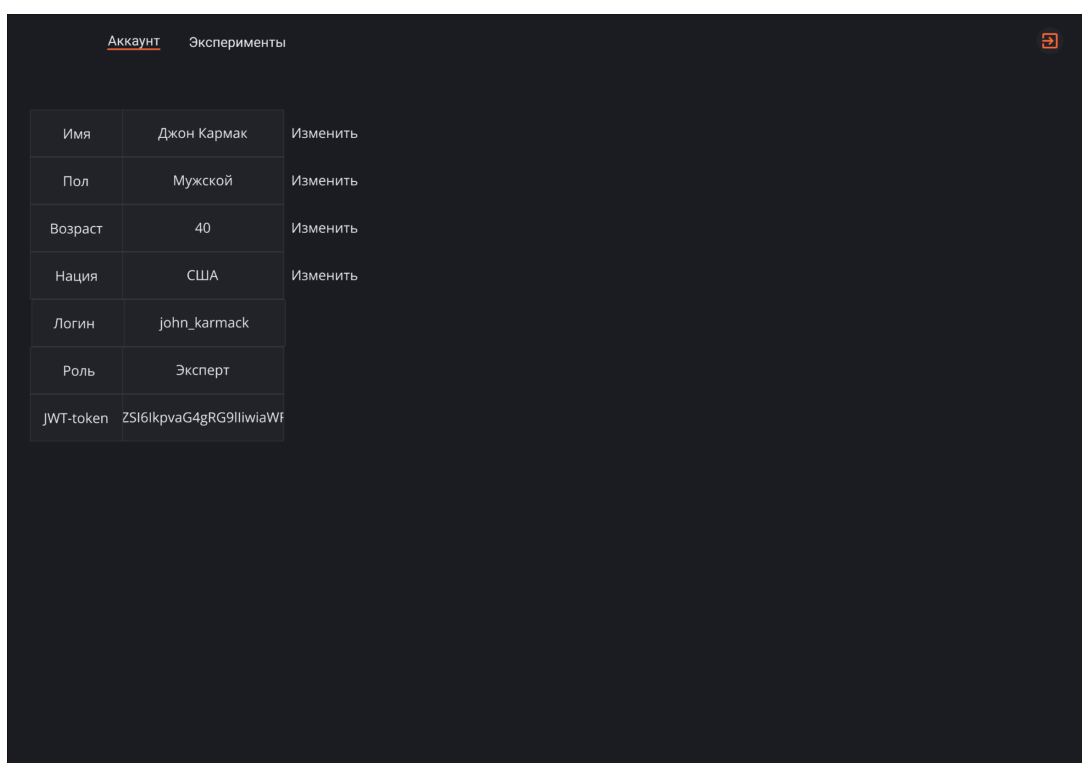


Рисунок 11 – Страница с информацией о пользователе

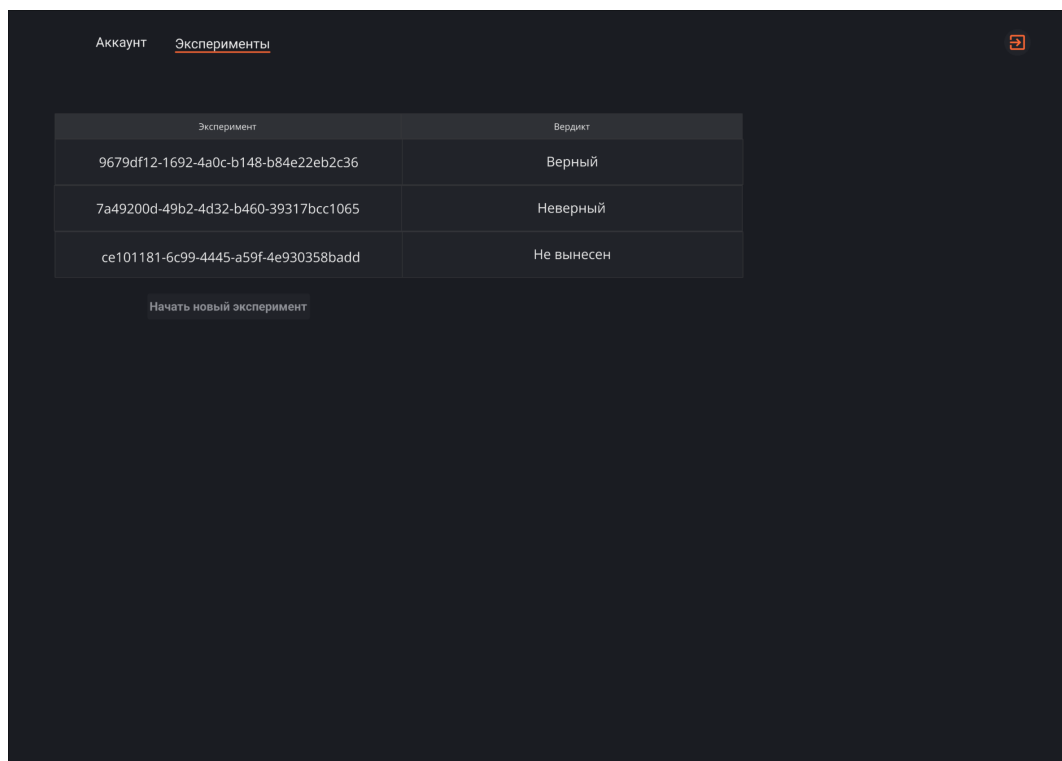


Рисунок 12 – Страница с доступными экспериментами

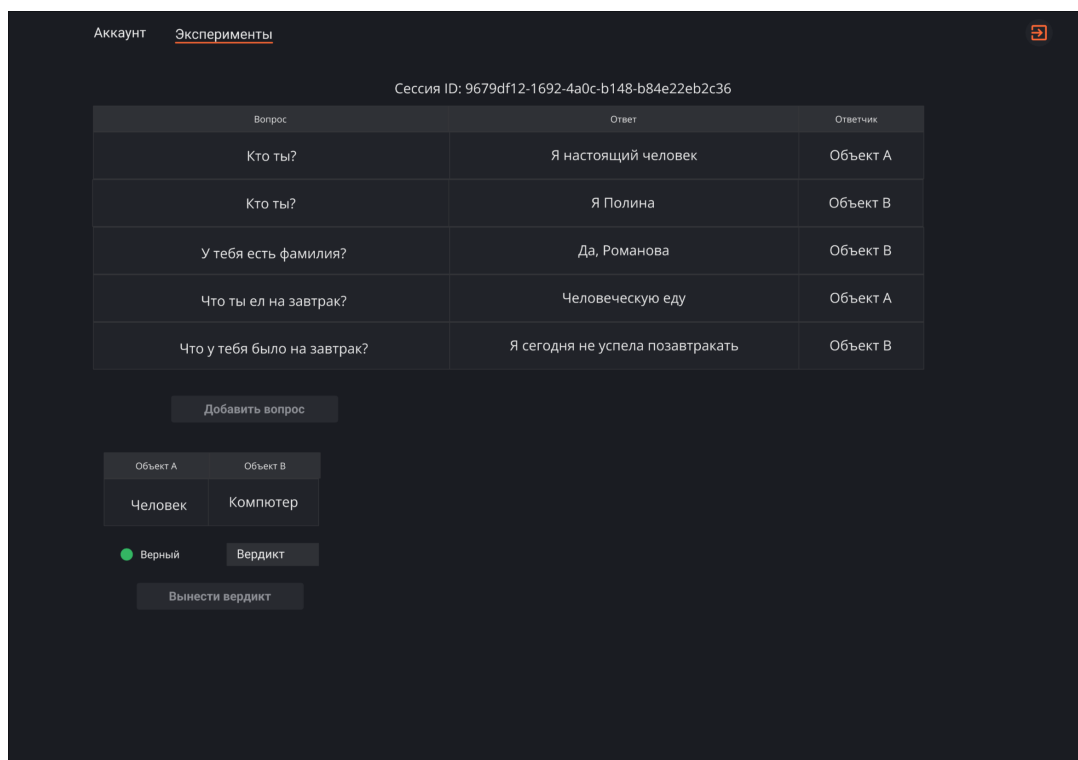


Рисунок 13 – Страница с экспериментом

Вывод

В данном разделе были представлена средства реализации программно-го обеспечения, листинги ключевых компонентов системы и пример работы приложения.

4 Исследовательский раздел

В данном разделе приведены описание исследования и технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

4.1 Постановка задачи исследования

4.1.1 Цель исследования

Целью исследования является сравнение времени, требуемого для получения сильносвязанных данных о тесте Тьюринга в двух базах данных SurrealDB и PostgreSQL.

4.1.2 Описание исследования

Сравнить занимаемое время для получения данных для различных баз данных можно при помощи бенчмарков — специальных функций, которые проводят серии различных испытаний с записью производительности системы для дальнейшего их сравнения. Для SurrealDB в рамках бенчмаркинга были написаны запросы, утилизирующие её графовую составляющую. В то же время запросы к PostgreSQL применяют множественные JOIN-запросы ввиду сильносвязанности данных.

На листинге 5 приведён SQL-запрос для PostgreSQL, по которому проводилось сравнение. Следует отметить, что в случае PostgreSQL связь между таблицами осуществляется при помощи внешних ключей (Foreign Key). Во всем остальном, таблицы аналогичны тем, что написаны для SurrealDB.

Листинг 5 – SQL запрос для PostgreSQL.

```
SELECT
    Session.SessionID ,
    Interrogator.Name ,
    Computer.Model ,
    Human.Name ,
    Question.QuestionText ,
    Answer.AnswerText ,
    Verdict.Correct
FROM Session
    JOIN Interrogator ON Session.SessionID = Interrogator.SessionID
    JOIN Computer ON Session.SessionID = Computer.SessionID
    JOIN Human ON Session.SessionID = Human.SessionID
    JOIN Question ON Session.SessionID = Question.SessionID
    JOIN Answer ON Session.SessionID = Answer.SessionID
    JOIN Verdict ON Session.SessionID = Verdict.SessionID
```

Аналогичный запрос написанный для **SurrealDB**, утилизирующий графовую составляющую базы данных:

```
SELECT id,
    <-participateIn<-human.name ,
    <-participateIn<-interroagtor.name ,
    <-participateIn<-computer.model ,
    ->includes->answer.text ,
    ->includes->question.text ,
    ->includes->verdict.correct
    FROM session;
```

Для замера производительности двух различных баз данных при выполнении запросов использовались библиотека **Criterion**, функции которой использовались для определения эффективности запросов по времени. Количество итераций и алгоритм анализа выборок контролируются библиотекой [33] и работают следующим образом:

- Перед началом самих замеров проводится этап «разогрева». На этом этапе процедура выполняется повторно, чтобы дать операционной системе и процессору время адаптироваться к новой рабочей нагрузке.

- Измерения проводятся на нескольких выборках. Каждая выборка состоит из одной или нескольких итераций процедуры.

По мере выполнения измерений количество итераций выборки увеличивается. Количество итераций вычисляется следующим образом:

$$\text{Количество итераций} = [d, 2d, 3d, \dots Nd]$$

Где N - общее количество выборок, а d - коэффициент, рассчитанный на основе приблизительной оценки времени итерации, измеренной в течение периода разогрева, который используется для масштабирования количества итераций в соответствии с настроенным временем измерения — 60 секунд.

- Первым шагом в анализе является классификация выбросов. Каждая выборка классифицируется с использованием модифицированной версии метода Тьюки [34]. Во-первых, межквартильный диапазон (IQR) рассчитывается исходя из разницы между 25 - м и 75 - м процентилем. В методе Тьюки значения меньше (25 - й процентиль - $1,5 * \text{IQR}$) или больше ($75 - \text{й процентиль} + 1,5 * \text{IQR}$) считаются выбросами.

На втором шаге среди выборок вычисляется ряд метрик. В данном исследовании используется среднее значение полученных выборок.

4.1.3 Технические характеристики

Ниже приведены технические характеристики устройства, на котором будет проведено исследование:

- Операционная система: Arch Linux [35] 64-bit;
- Количество ядра: 4 физических и 8 логических ядер;
- Оперативная память: 16 Гб, DDR4;
- Процессор: 11th Gen Intel® Core™ i5-11320H @ 3.20 ГГц [36].

Во время тестирования устройство было подключено к сети электропитания и было нагружено только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.1.4 Результаты исследования

В таблицах 1 - 3 представлены результаты поставленного эксперимента, где сравнивается время исполнения в зависимости от количества сущностей в базе данных.

Таблица 1 – Результаты сравнения времени, для запросов к PostgreSQL и SurrealDB (количество элементов - 100 единиц)

Количество запросов	SurrealDB, мс	PostgreSQL, мс
1	58732	45812
5	255079	224391
10	527629	473282
25	1428489	1262168
100	5021948	4624102

Таблица 2 – Результаты сравнения времени, для запросов к PostgreSQL и SurrealDB (количество элементов - 1000 единиц)

Количество запросов	SurrealDB, мс	PostgreSQL, мс
1	63418	68719
5	284249	301548
10	577324	598925
25	1531132	1762836
100	6145253	7843628

Таблица 3 – Результаты сравнения времени, для запросов к PostgreSQL и SurrealDB (количество элементов - 5000 единиц)

Количество запросов	SurrealDB, мс	PostgreSQL, мс
1	76418	95213
5	328253	414436
10	663635	737435
25	1931132	2435168
100	7296236	9396236

На рисунках 14 - 16 представлены визуализация результатов поставленного эксперимента в виде графиков.

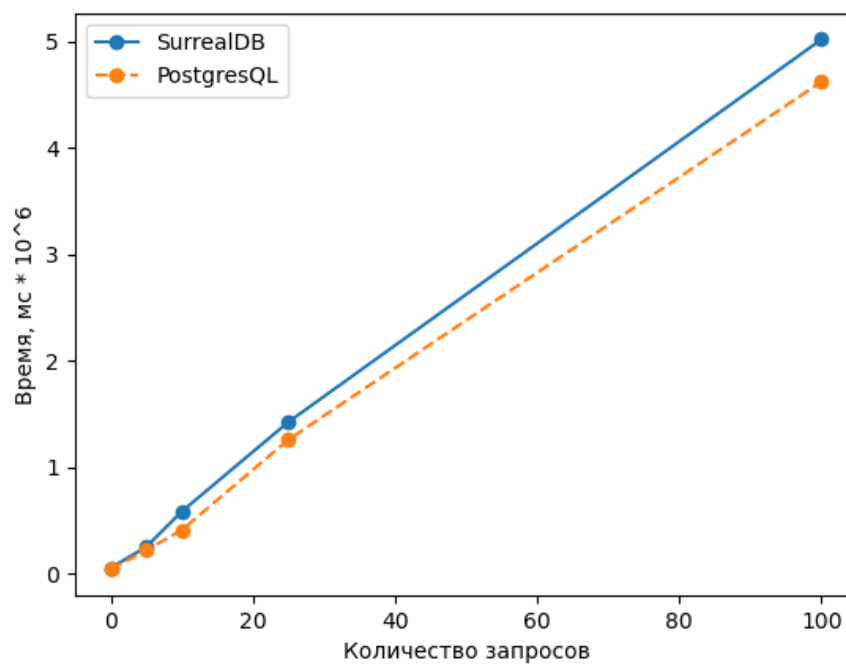


Рисунок 14 – Зависимость времени от количества запросов (количество элементов — 100)

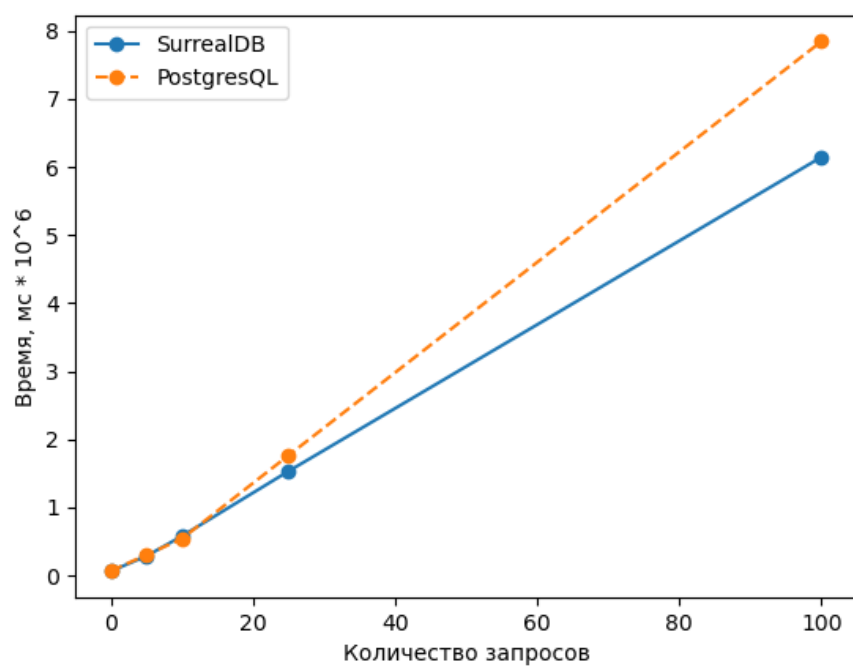


Рисунок 15 – Зависимость времени от количества запросов (количество элементов — 1000)

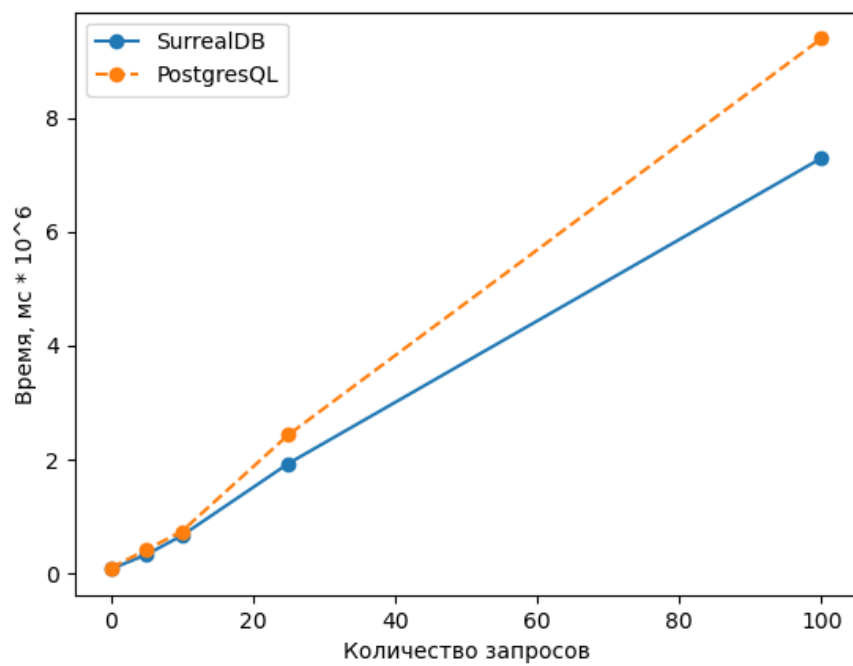


Рисунок 16 – Зависимость времени от количества запросов (количество элементов — 5000)

Вывод

Результаты эксперимента показали, что, в то время как при небольшом количестве сущностей в базе данных PostgreSQL показывает лучшие результаты, при увеличении количества сущностей в базе данных SurrealDB показывает лучшие результаты. Из данного наблюдения можно сделать следующий вывод: графовые базы данных показывают лучшие в сравнении с реляционными базами данных результаты при большом количестве сущностей в базе данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта, цель данного курсовой работы была достигнута, то есть был разработан программный продукт, позволяющий хранить результаты тестов Тьюринга.

Для достижение цели был выполнен ряд различных задач. Так, в первую очередь, был проведен анализ предметной области, определены основные сущности системы и связи между ними. Определена ролевая модель итогового приложения. Затем было проведено сравнение различных баз данных, был найден оптимальный вариант для решения поставленной задачи — **SurrealDB**. Была спроектирована база данных для хранения Тестов Тьюринга, для которой были определены 7 различных таблиц и 7 типов отношений. В качестве средства для реализации программного обеспечения был выбран язык **Rust** ввиду его быстродействия и удобства работы с ресурсами системы. Был разработаны серверная часть и графический интерфейс конечного приложения. Разработаны и проведены эксперименты по замеру времени работы программного обеспечения.

В ходе выполнения исследования было установлено, что запросы по связям могут выполняться быстрее в графовых базах данных, нежели запросы с использованием множественных JOIN-запросов в реляционных базах данных. При этом следует помнить, что значительные выигрыши по времени могут наблюдаться только при условии большого размера базы данных и сильной связанности сущностей внутри неё.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. What is natural language processing? [Электронный ресурс]. Режим доступа: <https://www.ibm.com/topics/natural-language-processing> (дата обращения: 01.04.2023).
2. Bourbakis Nikolaos G. Artificial Intelligence and Automation. World Scientific, 1998.
3. What are NoSQL databases? | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/topics/nosql-databases> (дата обращения: 09.04.2023).
4. Pavlo Andrew, Aslett Matthew. What's really new with NewSQL? // ACM SIGMOD Record. 2016. 09. Т. 45. С. 45–55.
5. UUID - MDN Web Docs Glossary - Mozilla [Электронный ресурс]. Режим доступа: <https://developer.mozilla.org/en-US/docs/Glossary/UUID> (дата обращения: 14.08.2023).
6. Oppy Graham, Dowe David. The Turing Test // The Stanford Encyclopedia of Philosophy / под ред. Edward N. Zalta. Metaphysics Research Lab, Stanford University, 2021.
7. TURING A. M. I.—COMPUTING MACHINERY AND INTELLIGENCE // Mind. 1950. 10. Т. LIX, № 236. С. 433–460. URL: <https://doi.org/10.1093/mind/LIX.236.433>.
8. Biessmann Felix, Treu Viktor. A Turing Test for Transparency // CoRR. 2021. Т. abs/2106.11394. URL: <https://arxiv.org/abs/2106.11394>.
9. Joublin Frank, Ceravola Antonello, Deigmoeller Joerg [и др.]. A Glimpse in ChatGPT Capabilities and its impact for AI research. 2023.

10. Zong Mingyu, Krishnamachari Bhaskar. a survey on GPT-3. 2022.
11. Chalmers David. GPT-3 and General Intelligence. 2020. URL: <https://dailynous.com/2020/07/30/philosophers-gpt-3/>.
12. Floridi Luciano, Chiriatti Massimo. GPT-3: Its Nature, Scope, Limits, and Consequences // Minds and Machines. 2020. 12. Т. 30. С. 1–14.
13. Kojima Takeshi, Gu Shixiang Shane, Reid Machel [и др.]. Large Language Models are Zero-Shot Reasoners. 2023.
14. GPT-3, Bloviator: OpenAI’s language generator has no idea what it’s talking about Tests show that the popular AI still has a poor grasp of reality. [Электронный ресурс]. Режим доступа: <https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/> (дата обращения: 14.08.2023).
15. The Turing Test: What Is It, What Can Pass It, and Limitations [Электронный ресурс]. Режим доступа: <https://www.investopedia.com/terms/t/turing-test.asp> (дата обращения: 12.08.2023).
16. Fast In-Memory SQL Analytics on Relationships between Entities / Chunbin Lin, Benjamin Mandel, Yannis Papakonstantinou [и др.] // CoRR. 2016. Т. abs/1602.00033. URL: <http://arxiv.org/abs/1602.00033>.
17. Yoon Byoung-Ha, Kim Seon-Kyu, Kim Seon-Young. Use of graph database for the integration of heterogeneous biological data // Genomics & Informatics. 2017. Mar. Т. 15, № 1. с. 19–27.
18. Besta Maciej, Iff Patrick, Scheidl Florian [и др.]. Neural Graph Databases. 2022. URL: <https://arxiv.org/pdf/2209.09732.pdf>.

19. SurrealDB | Features [Электронный ресурс]. Режим доступа: <https://surrealdb.com/features> (дата обращения: 09.04.2023).
20. SurrealDB | Architecture [Электронный ресурс]. Режим доступа: <https://surrealdb.com/docs/introduction/architecture> (дата обращения: 09.04.2023).
21. TiKV | Overview [Электронный ресурс]. Режим доступа: <https://tikv.org/> (дата обращения: 09.04.2023).
22. Neo4j Graph Database & Analytics | Graph Database And Management System [Электронный ресурс]. Режим доступа: <https://neo4j.com/> (дата обращения: 09.04.2023).
23. Sikhinam Tharun. How does the performance of a graph database such as Neo4j compare to the performance of a relational database such as Postgres? [Электронный ресурс]. Режим доступа: <https://courses.cs.washington.edu/courses/csed516/20au/projects/p06.pdf> (дата обращения: 09.04.2023).
24. What is ArangoDB? | ArangoDB Documentation [Электронный ресурс]. Режим доступа: <https://docs.arangodb.com/3.11/introduction/about-arangodb/> (дата обращения: 09.04.2023).
25. SurrealDB Documentation | Datamodel Overview [Электронный ресурс]. Режим доступа: <https://surrealdb.com/docs/surrealql/datamodel> (дата обращения: 14.08.2023).
26. JSON Schema | Type-specific keywords [Электронный ресурс]. Режим доступа: <https://json-schema.org/understanding-json-schema/reference/type.html> (дата обращения: 14.08.2023).

27. Electron | Documentation [Электронный ресурс]. Режим доступа: <https://www.electronjs.org/docs/latest/> (дата обращения: 12.08.2023).
28. Rust [Электронный ресурс]. Режим доступа: <https://www.rust-lang.org/>. (дата обращения: 19.08.2023).
29. Rust Borrow Checker [Электронный ресурс]. <https://doc.rust-lang.org/book/ch04-02-references-and-borrowing.html>.
30. Rust vs C clang fastest performance [Электронный ресурс]. Режим доступа: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust.html> (дата обращения: 14.08.2023).
31. Rust vs C++ g++ fastest performance [Электронный ресурс]. Режим доступа: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-gpp.html> (дата обращения: 14.08.2023).
32. Tauri Architecture [Электронный ресурс]. Режим доступа: <https://tauri.app/v1/references/architecture/> (дата обращения: 19.08.2023).
33. Criterion.rs | Analysis Process [Электронный ресурс]. Режим доступа: <https://bheisler.github.io/criterion.rs/book/analysis.html> (дата обращения: 14.08.2023).
34. Criterion.rs | Analysis Process [Электронный ресурс]. Режим доступа: <https://www.itl.nist.gov/div898/handbook/prc/section4/prc471.html> (дата обращения: 14.08.2023).
35. Arch Linux [Электронный ресурс]. Режим доступа: <https://archlinux.org/>. (дата обращения: 10.09.2023).
36. Процессор Intel® Core™ i5-11320H [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/217183/>

intel-core-i511320h-processor-8m-cache-up-to-4-50-ghz-with-ipu.
html. (дата обращения: 19.08.2023).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 13 слайдов.