СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Преподавание SQL чаще всего опирается на готовые методички и упражнения, которые преподаватель составляет вручную. Курсов и учебников хватает, но собрать хорошую практическую базу мешают малое число заданий и трудоёмкость их подготовки. Преподаватель тратит силы на проектирование баз данных, придумывание разных сценариев и ручную проверку ответов. Студентам же не хватает адаптивных упражнений, и это замедляет обучение и снижает интерес.

Спрос на знание SQL только растёт. Аналитики, дата-инженеры и специалисты по машинному обучению должны уверенно владеть и базовыми запросами, и продвинутыми возможностями реляционных СУБД. При этом существующие платформы для практики либо не дают преподавателю собрать собственные задания, либо не показывают, как студенты с ними справляются. Из-за этого учебный процесс отрывается от запросов индустрии.

Поэтому растёт интерес к инструментам, которые берут на себя рутину преподавателя и дают студенту интерактивную, подстраивающуюся под него среду. Особенно многообещающе выглядят локальные большие языковые модели (LLM). Они генерируют разные задания по реальной структуре базы данных и проверяют решения, не выпуская учебные данные за пределы организации.

Цель работы — разработать веб-сервис, который автоматизирует создание и проверку заданий по SQL, показывает схемы баз данных в виде диаграмм и даёт преподавателю аналитику. Такой сервис облегчает работу и преподавателю, и студенту.

Первый раздел разбирает предметную область (востребованность SQL, роль больших языковых моделей в образовании, устройство локального инференса) и анализирует существующие платформы для практики SQL. Второй раздел фиксирует цель, объект и предмет работы и формулирует требования к системе и интерфейсу. Третий обосновывает выбор технологий. Четвёртый, описывает программную реализацию, от сравнительного эксперимента по стратегиям генерации вопросов до архитектуры сервисов и деталей каждого эндпоинта. Пятый показывает интерфейс глазами преподавателя и студента, а шестой посвящён тестированию и развёртыванию. Завершают работу заключение и список использованных источников.

1 ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

По оценкам, спрос на специалистов с навыками работы с данными будет расти (США: вакансий для дата-аналитиков — +36 % к 2033 г. по данным Бюро статистики труда США [1]). По данным опроса разработчиков 2023 г. 48,66 % разработчиков и аналитиков регулярно используют SQL в своей работе [2].

Анализ открытых вакансий, свидетельствует о растущем спросе на специалистов, владеющих SQL. Годовой рост вакансий с упоминанием SQL за год составил 46 % [3]. Компании из самых разных отраслей включают SQL в требования к аналитикам и разработчикам [3], поскольку ведущие компании собирают и анализируют огромные объёмы данных.

Таким образом, владение SQL входит в число ключевых аналитических навыков, необходимых для работы с данными [4].

В университете SQL обычно преподают в рамках курсов баз данных и анализа данных. Практические занятия, работа с реальными базами данных оказываются наиболее эффективными. Исследования показывают, что обучение на практике – лучший способ освоения SQL [5].

Искусственный интеллект всё активнее применяется для автоматизации задач, которые раньше были доступны только человеку. Особенно ярко это проявляется в сфере IT-образования. Большие языковые модели (LLM) дают новые возможности для создания учебных материалов, проверки заданий и анализа прогресса студентов. Их способность обрабатывать информацию, включая схемы баз данных, позволяет создавать учебные планы, адаптированные под уровень подготовки учащихся.

Возможность использования языковых моделей локально, без обращения к облачным провайдерам, обеспечивает не только снижение затрат на инфраструктуру, при наличии необходимых мощностей, но и повышает безопасность, данные остаются в пределах внутренней сети, что критически важно для образовательных учреждений.

Успешные кейсы, такие как интеграция YandexGPT в учебную платформу “Яндекс Практикум”, или использование GPT-4 на платформе Khan Academy, подтверждают эффективность ИИ в образовании.

Однако, большинство существующих решений фокусируются на RAG системах для дополнительного объяснения студенту материала. Проект, представленный в рамках этой дипломной работы, делает акцент именно на автоматизацию процесса создания материалов для тестирования уровня студентов. Для преподавателей создавать разнообразные тестовые задания это трудоемкий процесс. Разработка каждого вопроса требует ручного проектирования баз данных, формулировки условий и проверки корректности ответов. Это не только увеличивает нагрузку на преподавателей, но и ограничивает вариативность заданий, что снижает мотивацию студентов.

1.1 Большие языковые модели и локальный инференс

Большая языковая модель — это нейросеть, обученная предсказывать следующий токен в последовательности. Модель, которая достаточно хорошо угадывает продолжение текста, попутно обретает ряд эмерджентных свойств, осваивает грамматику, факты о мире, умение рассуждать и писать код. В основе почти всех современных моделей лежит архитектура трансформера [6]. Её ключевая идея – механизм внимания (attention), который при обработке каждого токена позволяет модели взвешенно смотреть на все остальные токены контекста, а не только на ближайших соседей.

Размер модели принято измерять числом обучаемых параметров (весов). У моделей, с которыми работают сегодня, их от единиц до сотен миллиардов, и обозначение вроде «8B» означает восемь миллиардов параметров. Чем больше параметров, тем, как правило, выше качество и тем дороже она обходится и при обучении, и при использовании. В данной работе важна вторая стадия, инференс, то есть запуск уже обученной модели на готовых весах. Обучение требует кластеров из сотен видеокарт и здесь не рассматривается, а вот инференс реально организовать на одной машине.

Сколько памяти нужно для инференса, оценить несложно. Если каждый вес хранится в формате fp16, то есть в двух байтах, модель на 8 миллиардов параметров займёт около 16 ГБ только под веса, плюс ещё память под контекст. Это уже за пределами большинства потребительских видеокарт. Чтобы уменьшить размер модели, применяют квантизацию, то есть хранение весов в пониженной точности, например в 8 или 4 битах вместо 16. Методы вроде GPTQ [7] и AWQ [8] делают это подбирая квантованные значения так, чтобы ответы модели менялись минимально. На практике 4-битная квантизация урезает размер вчетверо: те же 8B укладываются примерно в 5 ГБ при едва заметной потере качества. Поэтому модель qwen3:8b в проекте занимает около 5,2 ГБ, а не 16.

Запускать модель локально, когда есть облачные API, имеет смысл по нескольким причинам. Данные не покидают периметр организации, а для учебного заведения, работающего с персональными данными студентов, это часто решающий аргумент. Нет зависимости от внешнего сервиса, API не станет недоступным, цена не вырастет, лимиты не будут исчерпаны в самый неподходящий момент. Затраты при таком подходе предсказуемы, заплатив один раз за железо, дальше платишь только за электричество, а не за каждый сгенерированный токен, а при появлении новых, более совершенных моделей с открытыми весами, старую можно легко заменить.

1.1.1 Инференс на CPU

Распространено заблуждение, что без видеокарты языковую модель не запустить. На самом деле инференс прекрасно работает и на обычном процессоре, просто заметно медленнее. Главный инструмент здесь – проект llama.cpp [9], движок инференса на C++, который считает модель на CPU, используя опера-

тивную память вместо видеопамяти. Веса при этом хранятся в формате GGUF, изначально рассчитанном на квантованные модели и компактную раскладку в памяти.

Скорость инференса на процессоре упирается не столько в число ядер, сколько в пропускную способность памяти: на каждый сгенерированный токен модель обязана один раз прочитать все свои веса из RAM. Следовательно, чем сильнее квантована модель, тем быстрее она работает на CPU, ведь читать приходится меньше байт. 4-битная модель на 7-8 миллиардов параметров на современном настольном процессоре выдаёт порядка нескольких токенов в секунду. Для чат-бота в реальном времени это медленно, а для задачи генерации заданий для студентов, где вопрос генерируется не в интерактивном режиме, такая скорость вполне допустима.

Ollama [10], которая используется в данной работе, как раз надстройка над llama.cpp. Она сама определяет, есть ли GPU. При его отсутствии выполняет вычисления на процессоре, а при наличии переносит на видеокарту столько слоёв модели, сколько помещается в видеопамять, оставляя остальное на CPU. Этот гибридный режим, когда часть модели лежит на GPU, а часть в оперативной памяти, позволяет запускать модели чуть крупнее, чем помещаются в VRAM целиком.

SQL остаётся ключевым навыком в аналитике и образовательных технологиях. Несмотря на трудности обучения и нужду в новых инструментах, сочетание традиционных подходов и инновационных средств может повысить эффективность обучения SQL. Успешные практики уже демонстрируют, что вовлечение ИИ может ускорить получение навыка и сделать обучение более индивидуальным.

1.2 Анализ аналогичных продуктов

1.2.1 Сервис HackerRank для SQL

HackerRank — крупная платформа для отработки навыков программирования, у которой есть отдельный раздел с задачами по SQL. Её активно используют и сами разработчики для практики, и компании для технических собеседований. Задачи сгруппированы по темам и сложности, решение пишется и проверяется прямо в браузере. Интерфейс решения задачи показан на рисунке 1.1.

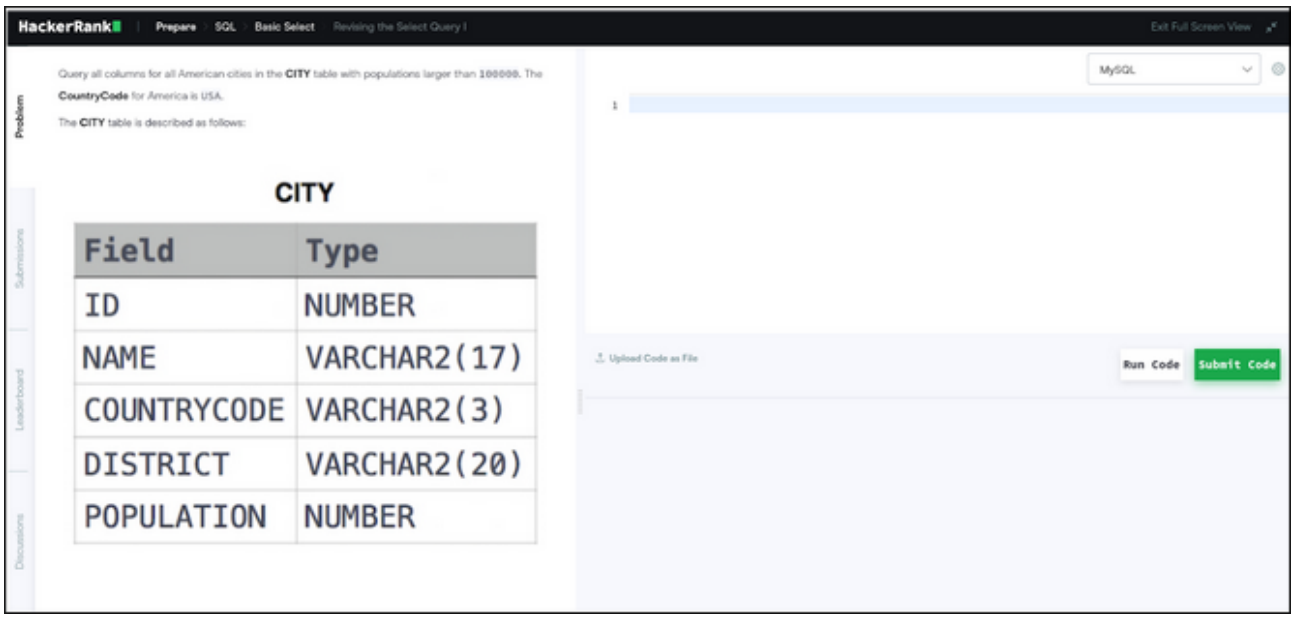

Рисунок 1.1 – Интерфейс решения задачи на сервисе HackerRank

- Сильные стороны: готовые задания, интеграция с корпоративными учебными программами.
- Ограничения: создание заданий преподавателем вручную, отсутствие автоматической визуализации данных и ER-диаграмм, невозможность отслеживания результатов студентов.

1.2.2 Сервис LeetCode для SQL

LeetCode известен прежде всего как площадка для подготовки к собеседованиям по алгоритмам, но у него есть и большой набор задач по базам данных. Сообщество вокруг платформы огромное, поэтому почти к каждой задаче есть обсуждения и разборы чужих решений. Запросы выполняются во встроенном редакторе и проверяются на готовых таблицах. Интерфейс платформы приведён на рисунке 1.2.

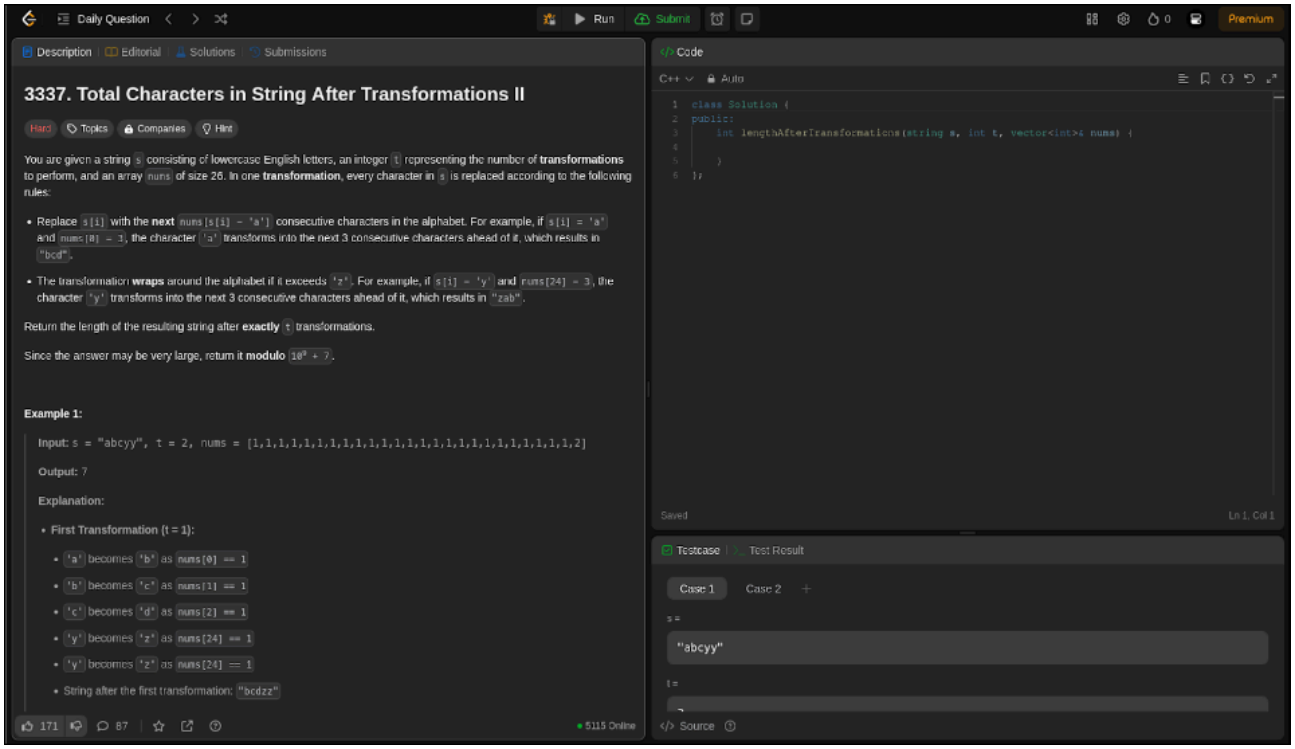

Рисунок 1.2 – Интерфейс решения задачи на сервисе LeetCode

- Сильные стороны: большой выбор заданий, популярность среди сообщества, удобный интерфейс.
- Ограничения: задания создаются вручную, нет ER-диаграмм, нет инструментов преподавателю для отслеживания прогресса студентов.

1.2.3 Тренажёр SQLZoo

SQLZoo — один из старейших бесплатных тренажёров по SQL, который многие вузы дают студентам как первое знакомство с языком. Уроки построены пошагово: короткая теория, затем серия упражнений прямо в браузере с мгновенной проверкой. Учебные базы небольшие и одни и те же для всех (классические примеры вроде таблицы стран мира), что удобно для самостоятельного старта, но не годится для работы с произвольной предметной областью. Интерфейс выполнения упражнения показан на рисунке 1.3.

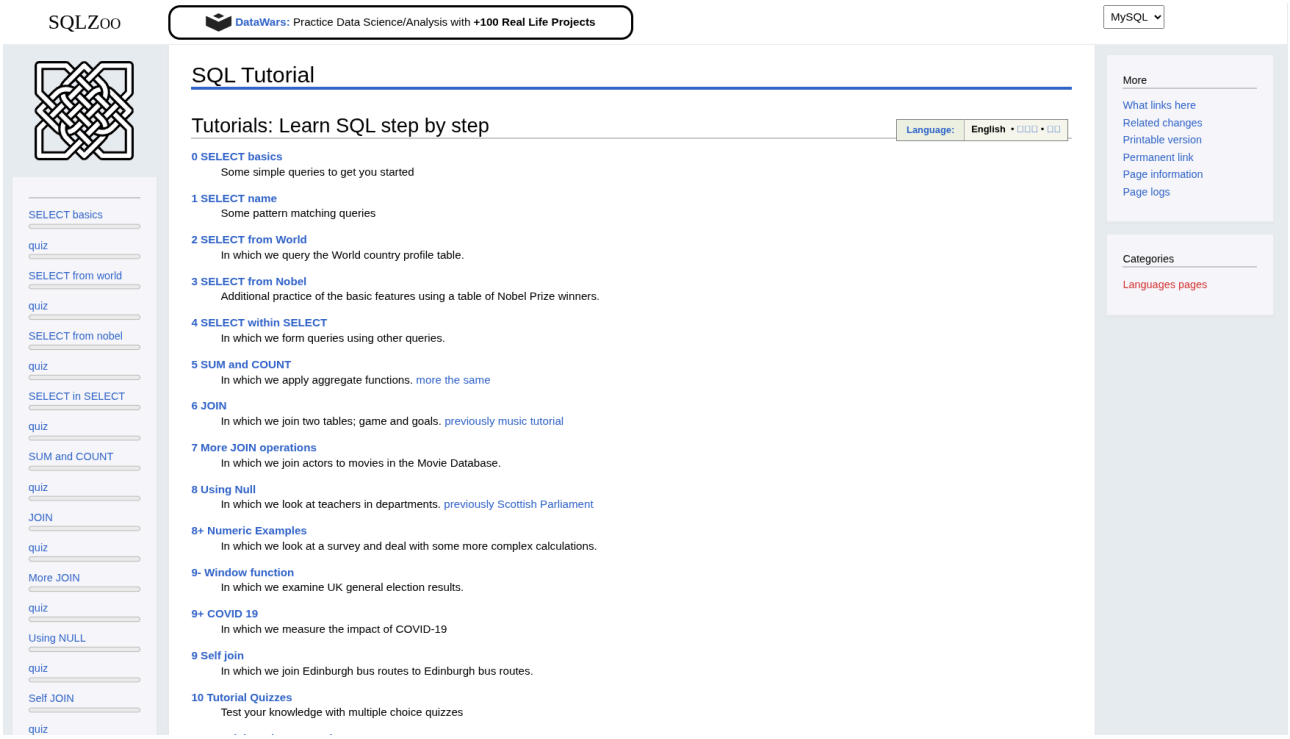

Рисунок 1.3 – Интерфейс выполнения упражнения на сервисе SQLZoo

- Сильные стороны: полностью бесплатный, низкий порог входа, проверка ответа сразу в браузере.
- Ограничения: фиксированный набор учебных баз, преподаватель не может загрузить свою БД или составить собственный тест, нет визуализации схемы и нет какой-либо аналитики по группе.

1.2.4 Платформа DataLemur

DataLemur нацелен на подготовку к собеседованиям на позиции аналитиков и дата-инженеров. Здесь собраны задачи, имитирующие реальные вопросы крупных компаний, с разбором решений и делением по темам. Контент качественный, но это закрытый банк задач, который формирует и пополняет сама платформа: ни преподаватель, ни студент не могут добавить туда задание на своей базе данных. Интерфейс платформы приведён на рисунке 1.4.

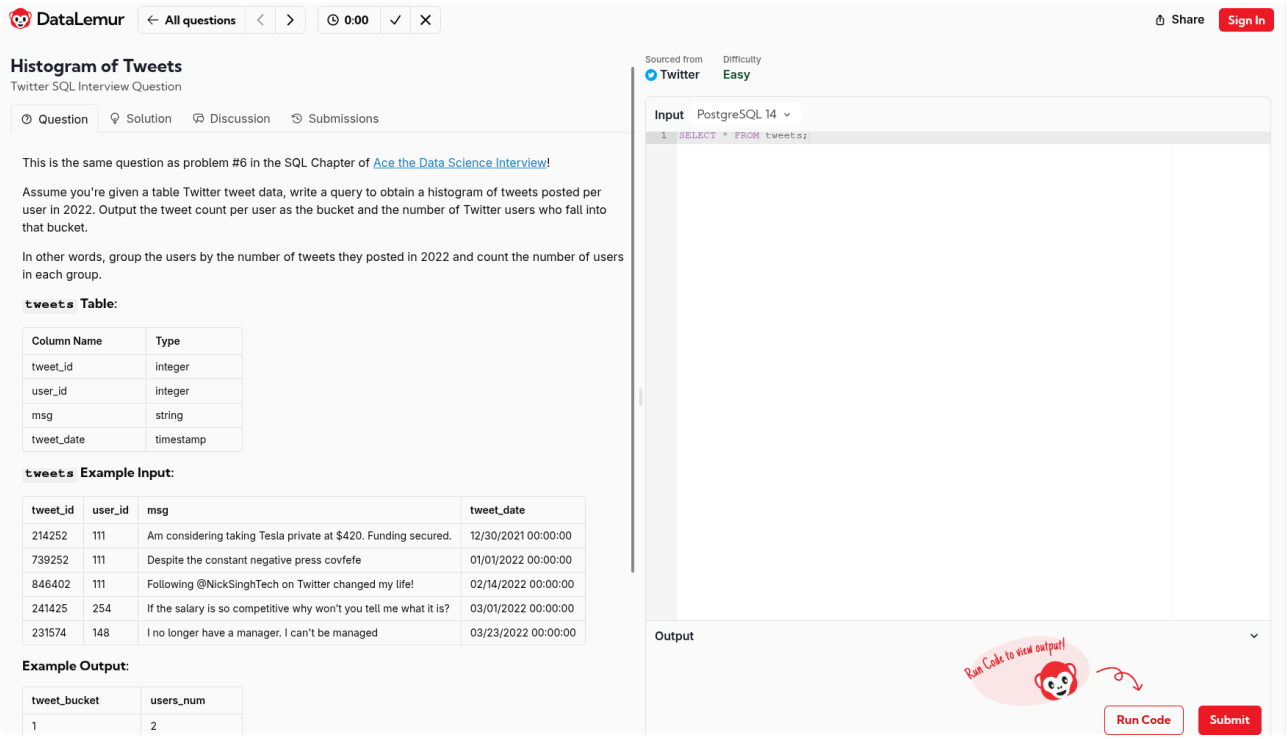

Рисунок 1.4 – Интерфейс решения задачи на платформе DataLemur

- Сильные стороны: задачи, приближенные к реальным собеседованиям, подробные разборы.
- Ограничения: узкая цель (подготовка к интервью), нет загрузки своих баз, нет режима преподавателя и отслеживания прогресса учебной группы.

1.2.5 PostgreSQL Exercises

PostgreSQL Exercises — бесплатный сайт с одной продуманной учебной базой (данные вымышленного спортивного клуба) и набором упражнений нарастающей сложности, от простых выборок до оконных функций. Проверка решения и эталонный ответ доступны прямо на странице задания. Это хороший справочный тренажёр, но он построен вокруг единственной схемы и не предполагает ни загрузки своих данных, ни роли преподавателя. Интерфейс упражнения показан на рисунке 1.5.

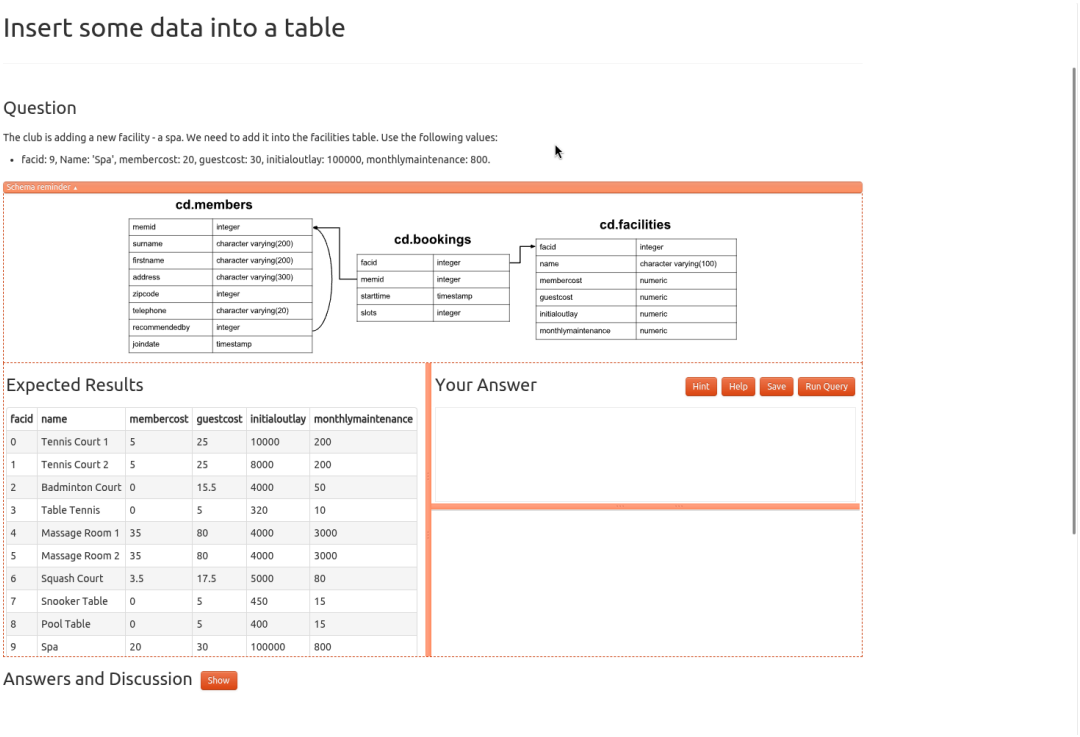

Рисунок 1.5 – Интерфейс упражнения на сайте PostgreSQL Exercises

- Сильные стороны: бесплатный, аккуратно подобранная прогрессия сложности, подходит для самопроверки.
- Ограничения: единственная фиксированная база, нет генерации заданий, нет ER-диаграмм и инструментов для группы.

1.2.6 Курс Codecademy «Learn SQL»

Codecademy предлагает интерактивный курс по SQL с встроенной средой выполнения запросов: студент читает объяснение и тут же пишет запрос, не выходя из браузера. Прогресс по курсу сохраняется. Формат линейный: это путь от урока к уроку, а не конструктор заданий, поэтому преподаватель не может собрать собственный тест на своей базе, а большая часть материалов доступна только по платной подписке. Интерактивный урок показан на рисунке 1.6.

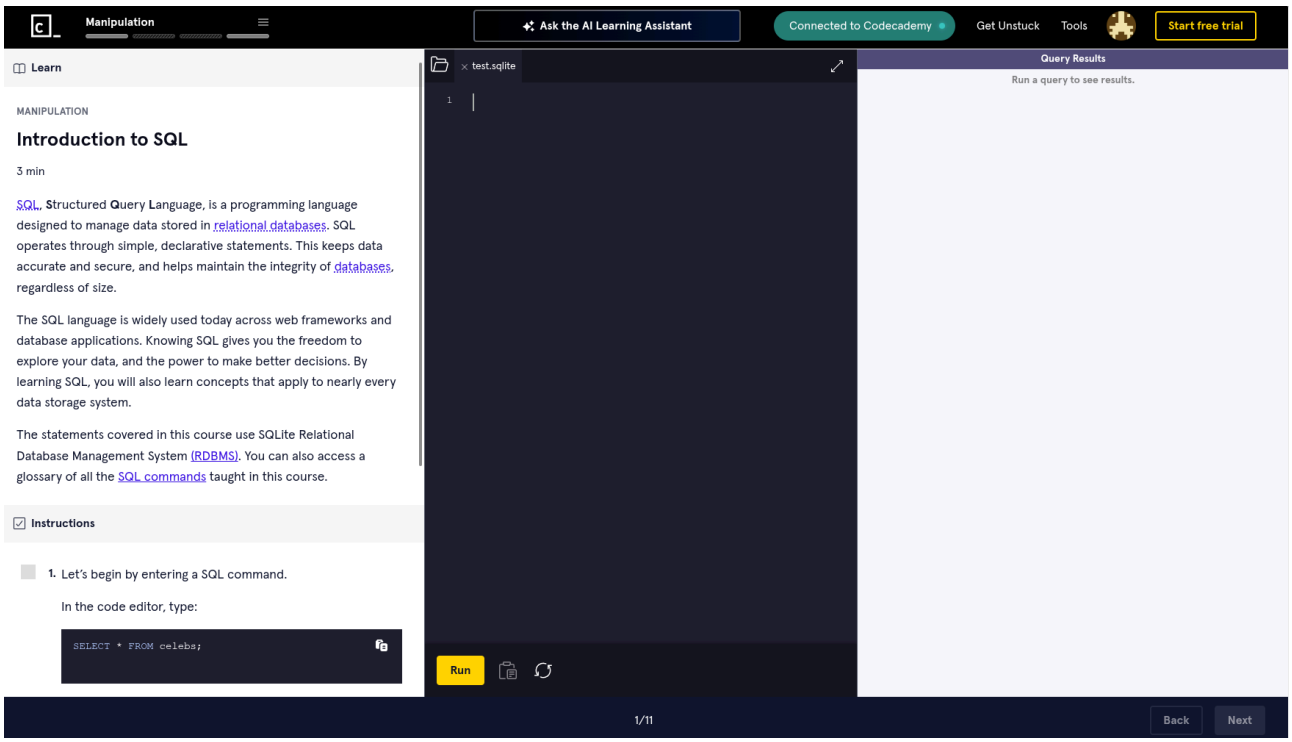

Рисунок 1.6 – Интерактивный урок SQL на платформе Codecademy

- Сильные стороны: интерактивная среда обучения, пошаговое сопровождение, сохранение прогресса.
- Ограничения: линейный курс вместо гибкого банка заданий, нет произвольных баз данных, основной контент платный, нет инструментов преподавателю.

Сравнение разобранных сервисов по признакам, которые важны именно для работы преподавателя, приведено в таблице 1.1.

Таблица 1.1 – Сравнение существующих сервисов с разрабатываемым решением

Сервис	**Своя БД**	**Генерация заданий**	**ER-диаграмма**	**Режим преподавателя**	**Аналитика по группе**	**Бесплатно**
HackerRank	–	–	–	–	–	±
LeetCode	–	–	–	–	–	±
SQLZoo	–	–	–	–	–	+
DataLemur	–	–	–	–	–	±
PostgreSQL Exercises	–	–	–	–	–	+
Codecademy	–	–	–	–	–	–
Разрабатываемый сервис	+	+	+	+	+	+

Существующие платформы предлагают удобный интерфейс и готовый контент, но они направлены в первую очередь на самообразование и формирование банка задач силами самой платформы или сообщества. Ни одно из решений не закрывает набор возможностей, нужный именно преподавателю: загрузить собственную базу данных, автоматически получить по ней задания нужного уровня

сложности, показать студенту схему в виде ER-диаграммы и собрать аналитику по группе.

Каждый сервис силён в своей нише (кто-то в подготовке к собеседованиям, кто-то в пошаговом обучении новичков), но все они оставляют создание заданий и контроль прогресса целиком на преподавателе. Разрабатываемый сервис заполняет именно этот разрыв. Вместо готового банка задач он даёт инструмент, который генерирует задания под любую загруженную базу и сразу даёт преподавателю картину результатов, а студентам предоставляет удобный интерфейс для проверки своих знаний.

2 ПОСТАНОВКА ЗАДАЧИ

Цель работы — разработка веб-сервиса для интерактивного обучения студентов языку SQL на практических задачах, который автоматизирует создание и проверку заданий, визуализирует схемы баз данных и даёт преподавателю инструменты аналитики.

Объект исследования — процесс практического обучения студентов языку SQL.

Предмет исследования — методы и средства автоматизации создания и проверки учебных SQL-заданий с применением локальных больших языковых моделей.

Для достижения цели поставлены следующие задачи:

1. проанализировать предметную область и существующие платформы для практики SQL, выявить их ограничения в контексте работы преподавателя;
2. сформулировать функциональные требования к системе и требования к пользовательскому интерфейсу;
3. выбрать и обосновать стек технологий для всех частей сервиса;
4. сравнить стратегии промптинга и языковые модели для генерации заданий и выбрать рабочую комбинацию;
5. спроектировать архитектуру и реализовать серверную часть: выполнение SQL-запросов, генерацию ER-диаграмм, извлечение схемы базы данных и генерацию вопросов через LLM;
6. реализовать клиентскую часть с разделением ролей преподавателя и студента;
7. протестировать систему и подготовить её к развёртыванию.

2.1 Функциональные требования к системе

Задачей выпускной квалификационной работы являлась разработка веб-сервиса для интерактивного обучения студентов языку SQL на практических задачах.

Система должна обеспечивать выполнение следующих функций:

1. Работа с базами данных: возможность загрузки файлов баз данных в формате SQLite, автоматическое извлечение и визуализация схемы данных в виде ER-диаграмм, получение текстового представления схемы для анализа LLM.
2. Автоматизация создания заданий: интеграция с большими языковыми моделями для автоматической генерации вопросов на основе анализа схемы базы данных, настройка уровней сложности заданий.
3. Проверка решений: автоматическое выполнение SQL-запросов студента на загруженной базе данных и сравнение результатов с верными ответами.
4. Система управления тестированием: интерфейс для преподавателей для создания и настройки тестов, выбора баз данных, модуль отслеживания

прогресса студентов, формирование отчетов о результатах прохождения каждого теста.

2.2 Требования к интерфейсу пользователя

Интерфейс системы должен быть разделен на две основные роли с соответствующим функционалом:

Для преподавателя:

1. Личный кабинет с возможностью загрузки баз данных, управления тестами и просмотра аналитики.
2. Инструменты создания вопросов вручную или автоматически с настройкой сложности.
3. Просмотр статистики прохождения тестов студентами (таблицы результатов и диаграммы распределения оценок).

Для студента:

1. Интерактивный редактор SQL-запросов с подсветкой синтаксиса.
2. Визуализация структуры базы данных (ER-диаграмма) непосредственно в интерфейсе решения задания.
3. Возможность получения мгновенной обратной связи по результатам выполнения запроса.
4. Возможность отправки ответа и завершения тестирования с сохранением результата.

3 ВЫБОР СРЕДСТВ РАЗРАБОТКИ

3.1 Система управления базами данных

Данные пользователей и схемы баз данных хранит Supabase [11] — открытая Backend-as-a-Service платформа на основе PostgreSQL. Supabase даёт RESTful и realtime API поверх PostgreSQL, встроенную аутентификацию и авторизацию, S3-совместимое хранилище файлов и удобную панель управления. В проекте S3 хранилище держит SQLite-базы, которые преподаватели загружают для своих тестов.

В пользу выбора Supabase есть несколько причин:

1. Supabase автоматически строит RESTful и realtime API по схеме PostgreSQL и сразу даёт эндпоинты для каждой таблицы и представления.
2. Встроенная аутентификация поддерживает OAuth-провайдеров, магические ссылки и роли, поэтому доступ преподавателей и студентов настраивается гибко [11].
3. Панель управления позволяет визуально редактировать схемы, настраивать RLS-политики доступа и следить за запросами.
4. Официальные клиентские библиотеки для JavaScript, Python и других языков дают удобный ORM-подобный интерфейс к данным.
5. Встроенное S3-совместимое хранилище (Supabase Storage) держит загруженные преподавателями SQLite-базы.

Открытый код позволяет развернуть Supabase на собственном сервере и полностью контролировать данные и инфраструктуру. Для учебного заведения с его требованиями к персональным данным это часто важно. Есть и облачная версия, с которой можно быстро начать разработку, не настраивая сервер.

3.2 Серверная часть

Серверная часть работает на Python 3.13 и FastAPI [12] — асинхронном веб-фреймворке поверх Starlette [13] и Pydantic [14]. FastAPI сам генерирует интерактивную документацию в Swagger UI, поэтому фронтенд легко подключать, а готовые пути API удобно тестировать. Pydantic-модели строго проверяют входные данные и ответы и убирают часть ошибок при обмене данными.

Асинхронная обработка запросов держит хорошую пропускную способность при множестве одновременных соединений. Сервером выступает Uvicorn [15], который поддерживает autoreload и ускоряет локальную разработку.

3.3 Клиентская часть

Клиентская часть опирается на стандартные HTML, CSS и JavaScript, а поверх них работают фреймворк Vue.js [16] и сборщик Vite.

Такой подход даёт компонентную архитектуру, где каждый элемент интерфейса — отдельный Vue-компонент, и его легко переиспользовать. Реактивность

Vue сама отслеживает изменения состояния и обновляет интерфейс, не заставляя вручную трогать DOM.

3.4 Интеграция с большими языковыми моделями

Задания генерирует локальная модель, развёрнутая через Ollama [10] — этот проект локально запускает любую LLM с открытыми весами. API у Ollama совместим с REST API OpenAI [17], который стал отраслевым стандартом и поддерживается большинством инструментов для разработки приложений с LLM.

Такой подход позволил запустить модель qwen3:8b [18] объёмом около 5,2 ГБ на локальном оборудовании. Время ответа получается быстрым и предсказуемым, без зависимости от внешних сервисов.

Оркестрация вызовов модели и пост-обработка результатов организованы через Python-фреймворк LangChain [19]. Он связывает шаги в цепочку, от подготовки входных данных и вызова LLM до проверки структуры ответа.

LangChain заодно унифицирует работу с разными провайдерами. В работе модель работает локально через Ollama, но провайдера легко сменить на любого облачного (OpenAI, DeepSeek, Claude, Qwen, OpenRouter и т.д.).

3.5 Среда разработки

Основная среда разработки — Visual Studio Code с богатой экосистемой расширений для Python, Go и JavaScript. Встроенный терминал и система задач позволяют запускать тесты и скрипты, не переключая окна, а графический интерфейс Git упрощает работу с ветками и коммитами.

4 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

4.1 Оценка стратегий генерации вопросов

Сервис генерации вопросов принимает на вход текстовое описание схемы БД и уровень сложности, возвращает формулировку вопроса и SQL-ответ. Способ использования llm в этой задаче определяет как именно модель интерпретирует запрошенный уровень сложности: без явных инструкций одна и та же цифра «3» может означать что угодно, от тривиального SELECT до запроса с тремя JOIN и оконными функциями. Чтобы выбрать стратегию обоснованно, был проведён сравнительный эксперимент на трёх языковых моделях, доступных для инференса на потребительском оборудовании, с тремя стратегиями промптинга.

4.1.1 Определение уровней сложности

Центральный элемент это методология определения уровня сложности задания. Большинство образовательных платформ используют субъективные ярлыки вроде «начальный», «средний», «продвинутый», которые каждая модель трактует по-своему. Здесь уровни определены через конкретные SQL-конструкции, что даёт однозначное соответствие между числом и ожидаемой структурой запроса.

Уровень 1 — простая выборка. `SELECT` с перечислением столбцов или `SELECT *`, одна таблица, никаких условий фильтрации. Задание проверяет что студент понимает базовый синтаксис и умеет выбрать нужные поля. Пример: «Выведите названия и цены всех товаров».

Уровень 2 — фильтрация. Добавляется `WHERE` с одним простым условием: равенство (`=`), числовые сравнения (`>`, `<`, `>=`, `<=`), текстовый поиск через `LIKE`, проверка пустых значений через `IS NULL` / `IS NOT NULL`. Всё ещё одна таблица, никаких объединений. Пример: «Найдите студентов, поступивших после 2020 года» или «Выведите товары дороже 1000».

Уровень 3 — JOIN или агрегация. Граница — появление либо первого `JOIN`, либо первой агрегатной функции. `JOIN` объединяет две таблицы по ключу и требует понимания реляционных связей. Агрегатные функции `COUNT`, `SUM`, `AVG`, `MIN`, `MAX` сворачивают множество строк в одно значение. `GROUP BY` допустим, но без `HAVING`. Важно что здесь один из двух инструментов, не оба вместе. Пример для JOIN: «Выведите названия курсов и имена преподавателей». Пример для агрегации: «Посчитайте количество заказов для каждого пользователя».

Уровень 4 — составные запросы. Несколько `JOIN` в одном запросе, или `GROUP BY + HAVING`, или один подзапрос через `IN` или `EXISTS`. Студент должен соединять три и более таблицы, фильтровать по агрегату или использовать вложенный `SELECT`. Пример с `HAVING`: «Найдите пользователей, сделавших более трёх заказов». Пример с подзапросом: «Выведите товары, которых нет ни в одном заказе».

Уровень 5 — продвинутые конструкции. Оконные функции (`OVER`, `ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LAG`, `LEAD`, `NTILE`, `FIRST_VALUE`), общие табличные выра-

жения через `WITH ... AS` (CTE), коррелированные подзапросы, операции над множествами `UNION` / `INTERSECT` / `EXCEPT`, или несколько уровней вложенности. Пример: «Для каждой категории найдите три самых дорогих товара с указанием их ранга». Такое задание почти неизбежно требует `ROW_NUMBER() OVER (PARTITION BY ...)`.

4.1.2 Стратегии генерации вопросов

Три стратегии отличаются тем, сколько информации о системе уровней сложности получает модель [20].

Zero-shot — базовая линия. Промпт содержит роль, схему БД и просьбу сгенерировать задание уровня N. Модель трактует уровень сложности на основе внутренних знаний [21]. Проблема: разные модели по-разному понимают числовые уровни, и без явного описания уровней калибровка непредсказуема.

Structured — к промпту добавляется таблица с описанием каждого уровня от 1 до 5: прописаны ожидаемые SQL-конструкции в точности как описано выше. Модель получает однозначное определение уровня перед генерацией.

Few-shot — к описанию уровней добавляются два готовых примера «вопрос + SQL» для запрошенного уровня. Примеры используют синтетическую схему, которая не входит в тестовый датасет: это исключает простое копирование структуры из примера, модель должна переносить паттерн на новую схему.

4.1.3 Автоматическая оценка качества генерации

Оценить сгенерированный текст труднее, чем его сгенерировать. Для задач с единственным правильным ответом всё просто, можно сравнить ответ модели с эталонным. Но у «хорошего учебного вопроса по SQL» эталона нет, удачных формулировок может быть бесконечно много. Классические метрики вроде BLEU и ROUGE, считают пересечение n-грамм с референсным текстом и для открытой генерации почти бесполезны: они штрафуют за любую перефразировку и ничего не знают о смысле сказанного.

Подход, который закрепился за последние пару лет, называется LLM-as-a-judge: качество генерации оценивает другая языковая модель, обычно более сильная. Расчёт на то, что модель уровня GPT-4 способна прочитать вопрос вместе с критериями и выставить оценку примерно так же, как сделал бы человек-эксперт. Работа, которая ввела этот подход в широкий оборот, показала высокую согласованность оценок модели-судьи с оценками людей на бенчмарке MT-Bench [22]. Развитие идеи — метод G-Eval [23], где судье дают не просто просьбу «оцени», а подробную инструкцию с описанием критерия и шкалы; это заметно повышает корреляцию с человеческой оценкой.

Судейство бывает двух видов. При попарном (pairwise) модели показывают два ответа и просят выбрать лучший. При поточечном (pointwise) показывают один ответ, который надо оценить по шкале. В бенчмарке далее используется поточечная схема: каждый сгенерированный вопрос оценивается отдельно по четырём независимым критериям, каждый по шкале от 1 до 5.

4.1.4 Систематические смещения модели-судьи

У судьи есть свои слабости, и игнорировать их нельзя. Известно несколько устойчивых искажений [22]. Позиционное смещение (position bias) проявляется при попарном сравнении: модель склонна предпочитать ответ на определённой позиции независимо от содержания, и на это прямо указывает работа «Large Language Models are not Fair Evaluators» [24]. Смещение в сторону многословия (verbosity bias) — это привычка судьи завышать оценку длинным развёрнутым ответам даже тогда, когда лишняя длина ничего полезного не добавляет.

Самое опасное искажение — self-bias, склонность модели завышать оценку текстам, которые она сама сгенерировала. Дело в том, что модель узнаёт собственный стиль. Показано, что языковые модели умеют отличать свой текст от чужого и систематически ставят ему оценки выше [25]. Если бы вопросы генерировала и оценивала одна и та же модель, её оценки были бы завышены в собственную пользу, а сравнение стратегий потеряло бы всякий смысл. Масштаб этого эффекта научились даже измерять отдельными статистическими методами [26].

Из этих ограничений и выросли решения, принятые в бенчмарке. Судьёй выбрана сильнейшая на текущий момент модель claude-opus-4.8, которая не участвует в генерации: так self-bias убирается подчистую, потому что ни один из оцениваемых вопросов ею не написан. Оценка идёт поточечно, по фиксированной рубрике с явным описанием каждого критерия и шкалы, в духе G-Eval, что снижает разброс между прогонами. А раз сравнения «ответ против ответа» нет вовсе, позиционное смещение в такой схеме просто не возникает.

4.1.5 Тестовые базы данных и метрики

Тест выполнялся на трёх базах данных из разных предметных областей: `store.db` (2 таблицы: products, categories), `university.db` (4 таблицы: students, courses, enrollments, professors), `ecommerce.db` (5 таблиц: users, orders, order_items, products, categories). Разное число таблиц принципиально: некоторые конструкции уровня 4-5 физически недостижимы на двухтабличной схеме, и тестировать только на ней было бы некорректно.

Каждое задание оценивалось двумя способами. Автоматические метрики: SQL выполняется на целевой базе данных, фиксируется корректность и наличие результата. Через библиотеку sqlglot разбирается структура запроса: число JOIN, подзапросов, оконных функций, наличие CTE, HAVING и т.д. Из взвешенной суммы этих признаков вычисляется `computed_complexity`, фактическая оценка сложности в диапазоне [1.0, 5.0].

Вторая группа метрик — оценки модели-судьи по описанной выше схеме LLM-as-a-judge. Каждое задание `claude-opus-4.8` оценивает по четырём критериям от 1 до 5: `clarity` (понятность вопроса для студента), `sql_alignment` (соответствие SQL вопросу), `level_match` (соответствие реальной сложности запрошенному уровню), `educational_value` (учебная ценность задания). Первичная метрика для сравнения стратегий — `level_match`: именно точная калибровка уровней является ключевой задачей.

4.1.6 Результаты

Сводные результаты по всем девяти комбинациям модели и стратегии собраны в таблице 4.1. Полужирным выделена итоговая комбинация — Qwen 3.5 9B со стратегией structured. Далее каждая метрика разбирается отдельно на графиках.

Таблица 4.1 – Сводные метрики бенчмарка по моделям и стратегиям

Модель	**Стратегия**	**SQL ok, %**	**Данные, %**	**clarity**	**sql_align**	**level_match**	**edu_value**
gemma-3-12b	zero-shot	100	80,0	4,53	4,73	3,00	3,53
	structured	100	86,7	4,47	3,60	4,20	3,53
	few-shot	100	93,3	4,33	4,20	4,00	3,93
gpt-oss-20b	zero-shot	73,3	46,7	3,33	3,07	2,13	2,80
	structured	64,3	50,0	3,50	3,36	3,64	3,07
	few-shot	80,0	80,0	3,80	3,80	4,07	3,60
qwen3.5-9b	zero-shot	80,0	60,0	4,07	3,40	1,67	2,80
	structured	**93,3**	**66,7**	**4,67**	**4,20**	**4,67**	**4,13**
	few-shot	80,0	66,7	4,20	3,93	4,33	4,07

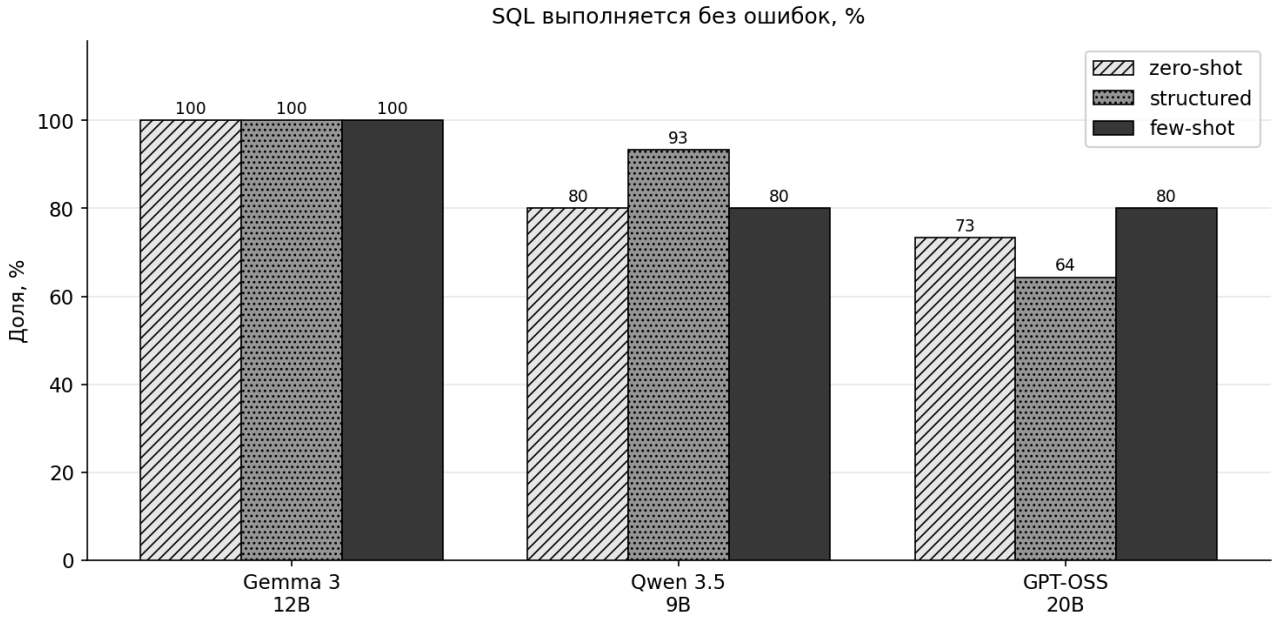

Рисунок 4.1 – Доля корректно выполненных SQL-запросов по моделям и стратегиям

Рисунок 4.2 – Доля запросов, возвращающих хотя бы одну строку данных

На Рисунок 4.1 и Рисунок 4.2 видно, что модели существенно расходятся по надёжности генерации. Gemma лидирует: 100% корректных SQL при любой стратегии, и стратегия на этот показатель вообще не влияет. Qwen показывает 80-93% в зависимости от стратегии, причём structured даёт максимум: модель лучше контролирует структуру вывода, когда явно знает, что от неё ожидается.

GPT-OSS-20b — самый ненадёжный кандидат: 64% валидных запросов при structured и только 47-50% запросов, которые возвращают хотя бы одну строку данных. Все его ошибки связаны с двумя причинами:

1. Модель генерирует настолько длинные внутренние рассуждения, что вывод обрывается по достижении лимита токенов и JSON не завершается корректно.
2. Модель игнорирует JSON-схему, по которой она должна дать ответ.

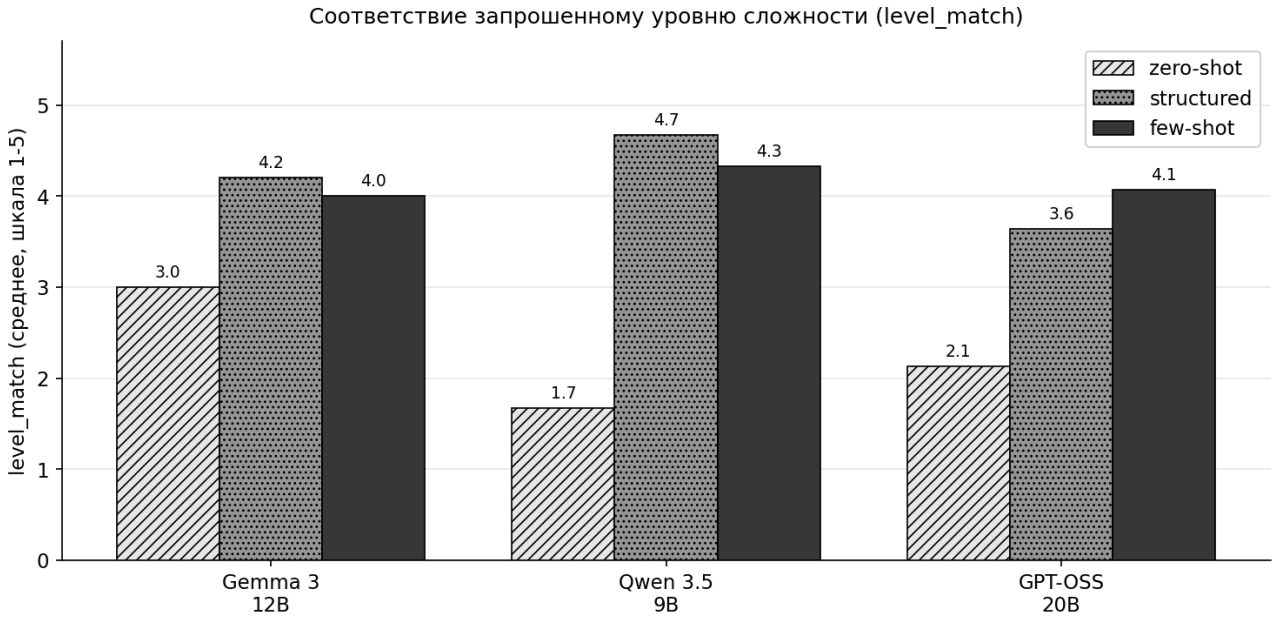

Рисунок 4.3 – Средний level_match (оценка судьи) по моделям и стратегиям промптинга

Самый выраженный эффект у Qwen [Рисунок 4.3]: zero-shot даёт средний level_match 1.67, structured — 4.67. Разрыв в три балла при неизменной модели достигается только за счёт изменения промпта. Gemma ведёт себя стабильнее: даже zero-shot даёт 3.0, structured 4.2. GPT-OSS-20b с учётом высокого процента ошибок менее надёжен как кандидат, хотя при few-shot достигает 4.07.

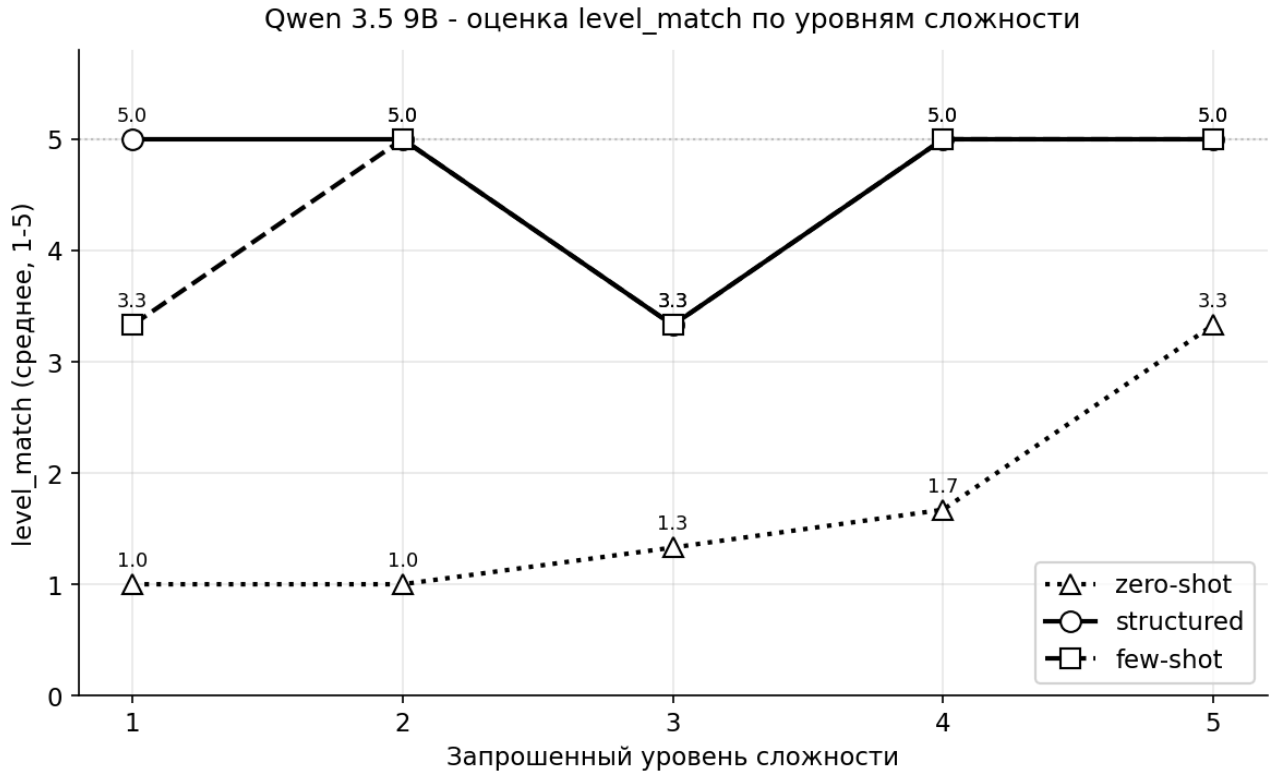

Рисунок 4.4 – level_match по запрошенному уровню сложности, Qwen 3.5 9B

Рисунок 4.4 показывает детализацию по уровням на примере Qwen, где разброс наиболее выраженный. При zero-shot модель стабильно выдаёт оценку 1 на уровнях 1-4: без явной классификации уровней модель вообще не различает их. После добавления structured или few-shot уровни 1, 2, 4 и 5 получают level_match 5.0. Уровень 3 немного ниже (3.33).

4.1.7 Оценка по остальным критериям судьи

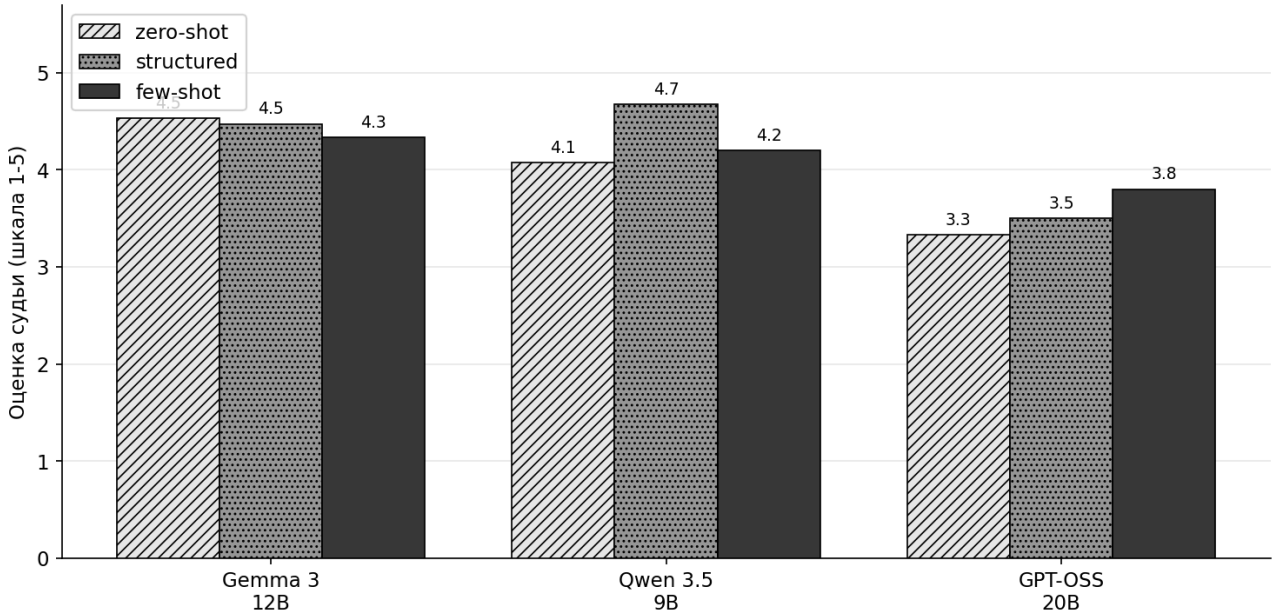

Рисунок 4.5 – Понятность формулировки вопроса для студента (clarity) по моделям и стратегиям

`Clarity` [Рисунок 4.5] стабильно высокий у всех трёх моделей вне зависимости от стратегии: диапазон 3.33-4.67. У Gemma zero-shot он даже максимален по всему тесту (4.53). Стратегия промптинга на читаемость вопроса почти не влияет: модели умеют формулировать задания понятно и без дополнительных инструкций. Это значит, что clarity не является дифференцирующим критерием при выборе стратегии: все три справляются с этой задачей.

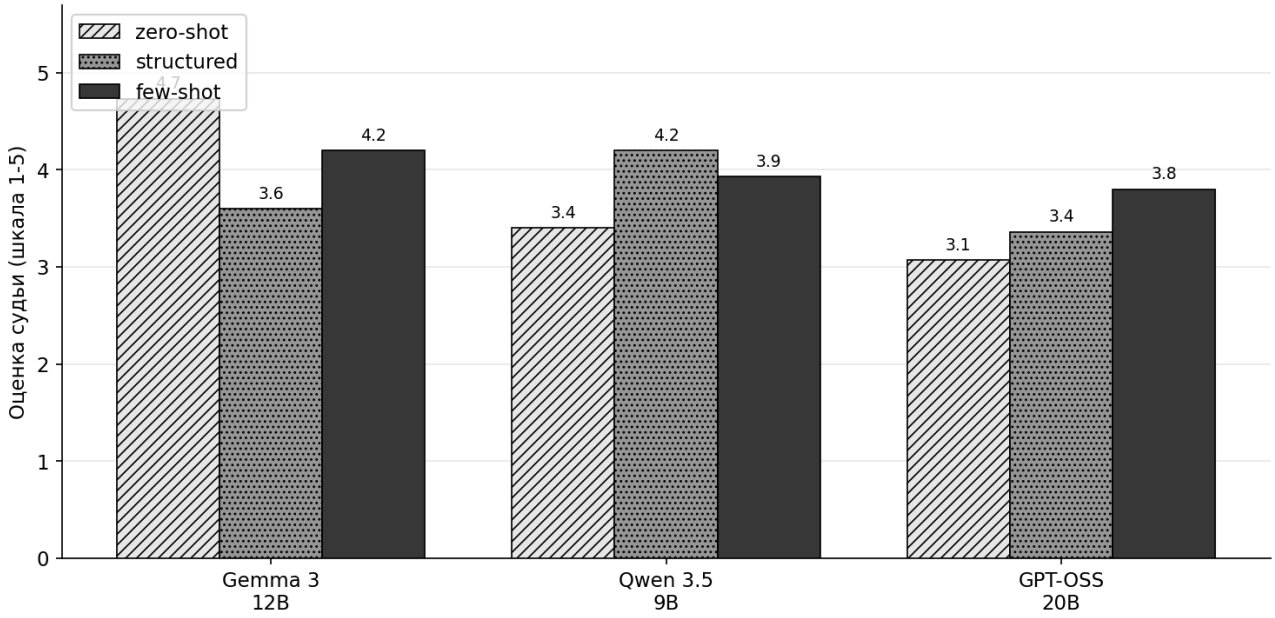

Рисунок 4.6 – Соответствие SQL вопросу (sql_alignment) по моделям и стратегиям

`Sql_alignment` [Рисунок 4.6] обнаруживает неочевидный эффект. Gemma при zero-shot набирает 4.73, наивысший показатель по всему тесту. SQL логически корректно отвечает на вопрос, хотя уровень сложности при этом неправильный. Ровно та же картина у Qwen zero-shot: sql_alignment 3.4 при level_match 1.67. Модель пишет корректный запрос, но не понимает, что от неё ожидают по сложности. При добавлении описания уровней sql_alignment не падает: у Qwen structured он составляет 4.2.

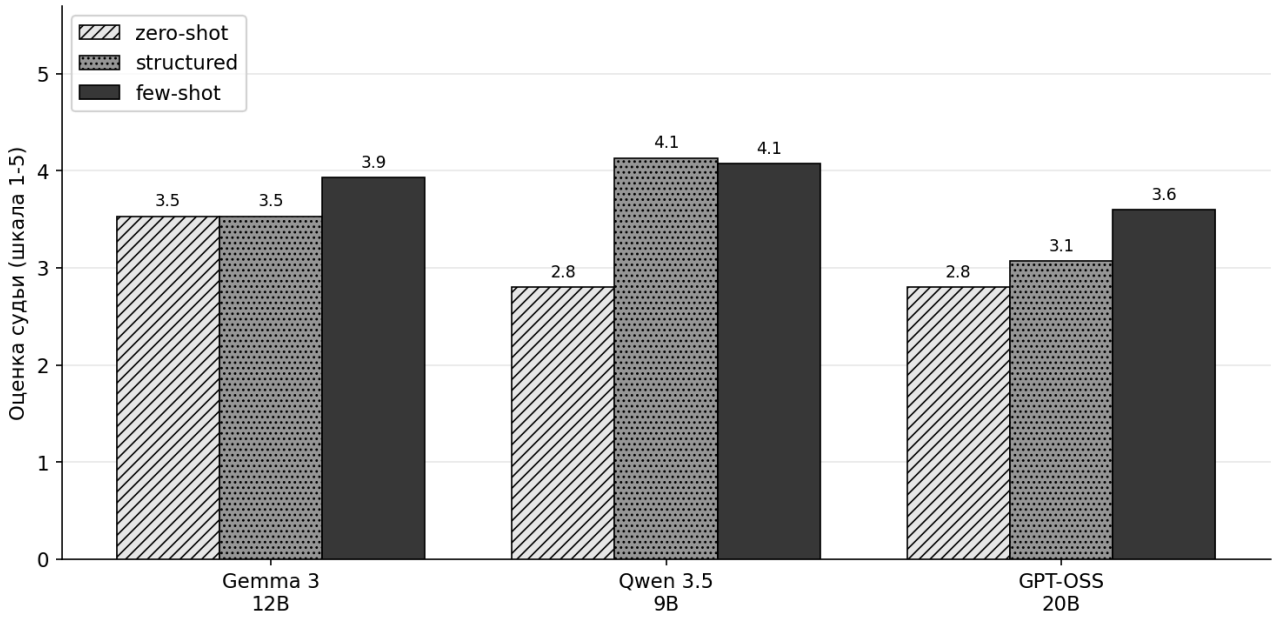

Рисунок 4.7 – Учебная ценность задания (educational_value) по моделям и стратегиям

`Educational_value` [Рисунок 4.7] ведёт себя синхронно с `level_match`, что закономерно. Когда задание попадает в нужный уровень, оно оказывается учебно ценным: Qwen structured и few-shot: 4.07-4.13. Когда промах по уровню, ценность

падает: Qwen zero-shot 2.8, GPT-OSS zero-shot 2.8. Задание уровня 5 с обычным SELECT без JOIN не даёт студенту ничего нового. Из всех метрик судьи именно educational_value наиболее тесно связан с тем, зачем вообще нужна точная калибровка уровней. По этому параметру Qwen лидирует при использовании стратегий Structured и Few-shot.

4.1.8 Выводы

Выбор для сервиса генерации вопросов — стратегия structured с моделью Qwen 3.5 9B. По совокупности критериев это лучший кандидат: level_match 4.67, clarity 4.67, educational_value 4.13. Structured предпочтительнее few-shot при равном качестве: эта стратегия тратит меньше токенов в каждом запросе как на вход так и на выход, что существенно снижает время на генерацию и стоимость в токенах или затрате вычислительных ресурсов. Дополнительно, при использовании этой стратегии, нет зависимости от актуальности примеров, то есть при изменении интерпретации цифр сложности не нужно менять примеры.

Модель Qwen 3.5 9B выигрывает в сравнении с Gemma 3 12B, так как она имеет меньше параметров и требует меньших ресурсов для локального инференса, без использования облачных провайдеров LLM, это позволяет развернуть ее на более дешевом оборудовании.

Бенчмарк показывает, что все три модели хорошо справляются с синтаксически корректным SQL и понятными формулировками, и это не дифференцирующий фактор. Ключевое различие только одно: способность правильно интерпретировать уровень сложности без явного его определения. Именно здесь выбор стратегии имеет значение.

4.2 Архитектура системы

Веб-сервис имеет трёхзвенную архитектуру и состоит из трёх основных компонентов: клиентская часть (Frontend), два микросервиса на FastAPI и облачная инфраструктура Supabase. Общая схема архитектуры приведена на рисунке 4.8.

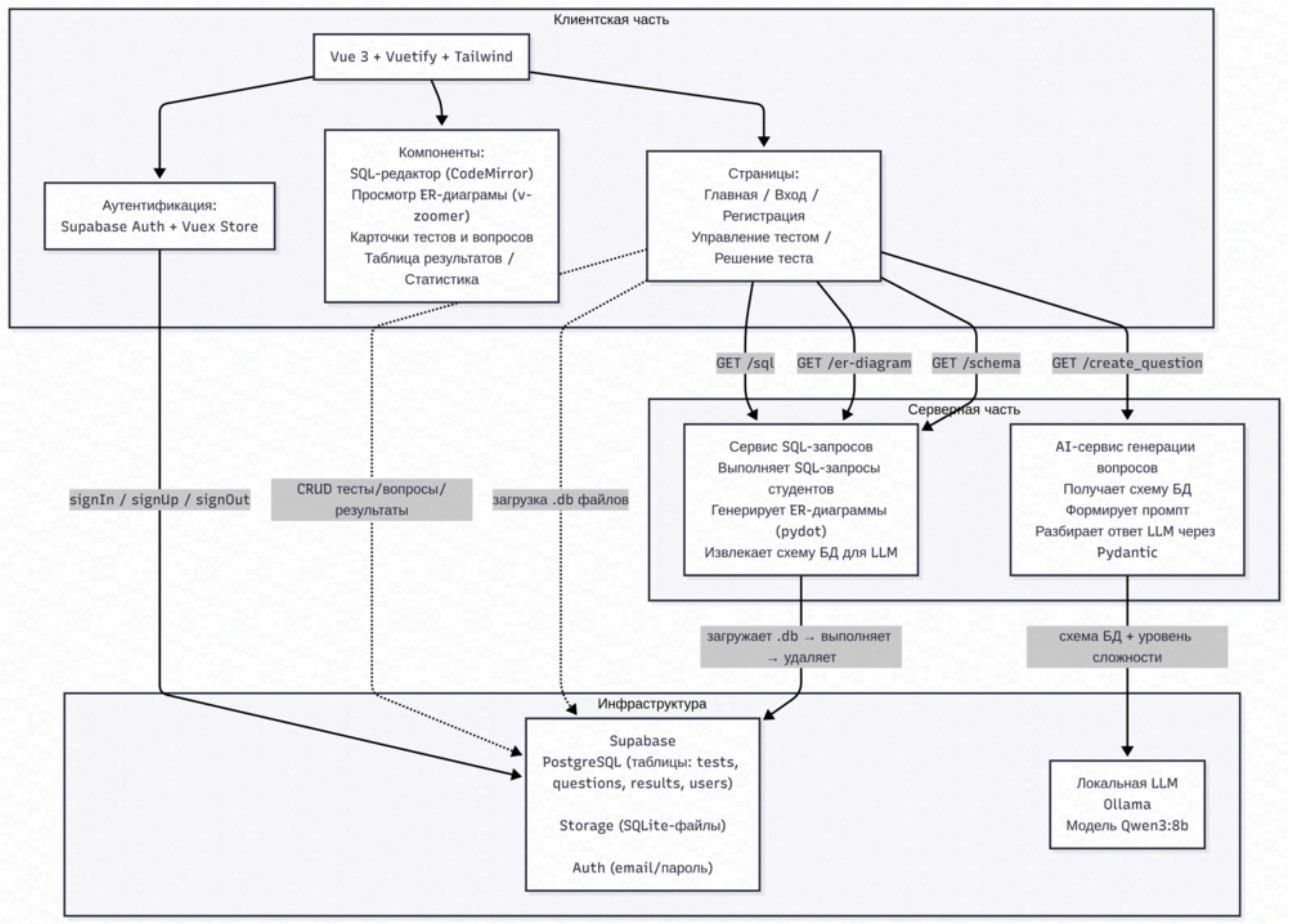

Рисунок 4.8 – Архитектура веб-приложения

Клиентская часть (Frontend) реализована на веб фреймворке Vue3. В современной веб-разработке чистый CSS используется редко, поэтому в данной работе применена современная библиотека для вёрстки — Tailwind CSS [27]. Ключевая особенность этой библиотеки — возможность задавать классические CSS-свойства прямо в атрибуте class HTML-тега. Такой подход заметно ускоряет и упрощает разработку, особенно в связке с Vuetify [28].

Vuetify — это библиотека компонентов интерфейса, написанная специально для фреймворка Vue.js. Она предоставляет обширный набор компонентов, таких как модальные окна, кнопки, поля ввода, контейнеры, карточки и т.д. Этот набор компонентов строго соблюдает все правила и гайдлайны от Google в дизайн системе Material 3 [29], что позволяет сконцентрироваться на функционале, а не создании собственной дизайн-системы.

Серверная часть (Backend) разделена на два микросервиса для изоляции ответственности и возможности независимого масштабирования при возрастании нагрузки на одну из частей сервиса:

1. SQLite Query Service — основной бэкенд для работы с базами данных. Он отвечает за обработку запросов и выполнение SQL-кода на базе данных для теста, извлечение схемы и генерацию ER-диаграмм. Сервис получает UUID теста, получает по ID теста путь к базе данных, загружает нужный файл .db

из Supabase S3 Storage во временный файл, выполняет операцию и удаляет файл.

2. AI Service — сервис для генерации вопросов. Получает текстовое описание схемы БД из SQLite Query Service и передаёт его в LLM через LangChain.

Инфраструктура Supabase используется как: база данных PostgreSQL, сервис аутентификации, файловое хранилище (Storage для SQLite-файлов), RESTful API с автоматической генерацией эндпоинтов.

4.2.1 Схема взаимодействия при выполнении ключевых сценариев

Сценарий 1. Создание теста.

Преподаватель заполняет форму с полями «Вопрос» и «Ответ», где «Вопрос» — это формулировка задачи для которой студент должен написать SQL запрос, а «Ответ» — это SQL запрос, который выполняет поставленную задачу.

Фронтенд загружает файл `.db` в Storage и получает ссылку на файл базы данных из Storage

URL файла сохраняется в поле `db_file_url`.

Фронтенд создаёт запись в таблице `tests` через API запрос к Supabase. Пример созданной записи показан на рисунке 4.9.

id uuid	title text	description text	teacher_id uuid	db_file_url text
b19c90c5-7ad8-4c01-b2e2-fb9d71d62973	Тест на JOIN	Тест проверяет умение использовать оп	1abe7b5a-372b-4958-a2c6-c6f95...	https://hlbwkzdazfqxhqylpwne.supabase

Рисунок 4.9 – Запись в таблице tests

Сценарий 2: Генерация вопроса через AI.

Преподаватель нажимает «Сгенерировать»

Фронтенд обращается к эндпоинту `/schema` в SQLite Query Service

Получает текстовую схему

Отправляет в `/create_question` из AI Service

LLM генерирует reasoning, формулировку вопроса и правильный ответ

Фронтенд отображает результат для подтверждения

Преподаватель сохраняет вопрос в Supabase.

Сценарий 3: Решение теста студентом.

Студент открывает страницу теста

Фронтенд получает список вопросов из Supabase и ER-диаграмму из SQLite Query Service

Студент пишет SQL-запрос в редакторе

Нажимает «Выполнить»

Запрос отправляется на `/sql` в SQLite Query Service

Результат отображается в таблице

При нажатии «Отправить ответ» результат запроса студента сравнивается с результатом эталонного запроса

После ответа на все вопросы результаты отправляются в таблицу `results` для дальнейшего отображения статистики в личном кабинете преподавателя

4.3 Детальная реализация SQLite Query Service

Сервис реализован на FastAPI. Он предоставляет три эндпоинта, каждый из которых работает по общему принципу: получение записи теста из Supabase, загрузка .db файла во временную директорию, выполнение операции, удаление временного файла.

4.3.1 Эндпоинт `/sql` выполнение SQL-запросов

Параметры: `record_id: str` (UUID теста), `query: str` (SQL-запрос).

Листинг 1. Функция выполнения SQL-запроса

```
@router.get("/sql")
async def execute_sql(record_id: str, query: str):
    file_name = None
    conn = None
    try:
        record = get_test_record(record_id)
        if not record:
            raise HTTPException(status_code=404)
        db_file_url = record.get("db_file_url")
        if not db_file_url:
            raise HTTPException(status_code=400)
        file_name = str(uuid.uuid4())
        download_db_file(db_file_url, file_name)
        conn = sqlite3.connect(file_name)
        cursor = conn.cursor()
        cursor.execute(query)
        result_data = cursor.fetchall()
        column_names = [desc[0] for desc in cursor.description]
        result = [
            {column: value for column, value in zip(column_names, row)}
            for row in result_data
        ]
        conn.close()
        os.remove(file_name)
        return {"columns": column_names, "result": result}
    except HTTPException:
        raise
    except Exception as e:
        if conn: conn.close()
        if file_name and os.path.exists(file_name):
            os.remove(file_name)
        raise HTTPException(status_code=500, detail=str(e))
```

Функция принимает идентификатор теста и текст SQL-запроса. Загружает соответствующую БД из Supabase Storage, выполняет запрос через sqlite3 [30], возвращает имена колонок и строки результата в формате JSON. Гарантирует очистку временного файла даже при ошибке.

4.3.2 Эндпоинт `/er-diagram` генерация ER-диаграммы

Параметры: `record_id: str`.

Возвращает ER-диаграмму в формате PNG, закодированную в Base64. Генерация выполняется библиотекой `pydot` [31]:

Листинг 2. Функция генерации ER-диаграммы

```
def generate_er_diagram(database_file):
    conn = sqlite3.connect(database_file)
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
    tables = cursor.fetchall()
    table_info = {}
    for table in tables:
        cursor.execute(f"PRAGMA table_info({table[0]});")
        columns = cursor.fetchall()
        cursor.execute(f"PRAGMA foreign_key_list({table[0]});")
        foreign_keys = cursor.fetchall()
        table_info[table[0]] = (columns, foreign_keys)
    graph = pydot.Dot(graph_type='graph', rankdir='LR')
    for table, (columns, foreign_keys) in table_info.items():
        node = pydot.Node(table, shape='record')
        label = f'{{ {table} | {{'
        for column in columns:
            label += f' {column[1]} : {column[2]}\\l'
        label += '} }}'
        node.set_label(label)
        graph.add_node(node)
        for fk in foreign_keys:
            arrowhead = 'crow' if table < fk[2] else 'tee'
            edge = pydot.Edge(
                f"{table}:{fk[3]}",
                f"{fk[2]}:{fk[4]}",
                arrowhead=arrowhead
            )
            graph.add_edge(edge)
    return graph.create(format='png')
```

Функция подключается к SQLite-файлу, извлекает метаданные о таблицах через `PRAGMA table_info` и `PRAGMA foreign_key_list`, строит граф с помощью pydot. Каждая таблица отображается как узел с перечислением колонок, внешние ключи — как рёбра с разными типами стрелок для указания направления связи.

4.3.3 Эндпоинт `/schema` получение схемы БД

Параметры: `record_id: str`.

Возвращает текстовое представление схемы базы данных, очищенное от лишних пробелов и модификаторов `NOT NULL`. Используется для передачи в AI Service при генерации вопросов.

4.3.4 Клиент Supabase (`supabase_client.py`)

Модуль предоставляет набор функций для взаимодействия с Supabase через HTTP-запросы:

- `get_test_record(record_id)` — получает запись теста из таблицы `tests`.

- `get_db_file_url(record_id, db_filename)` — получает URL по ID теста из таблицы `tests` для загрузки файла из Storage.
- `download_db_file(url, destination)` — скачивает файл БД во временную директорию.

4.4 Детальная реализация AI Service

Сервис предоставляет единый эндпоинт для генерации вопросов.

4.4.1 Эндпоинт `/create_question` — генерация вопроса

Параметры: `schema: str` (текст схемы базы данных sqlite), `complexity: str` (уровень сложности 1–5).

Механизм генерации:

1. Схема БД передаётся в LangChain-цепочку.
2. Модель LLM получает системный промпт с подробной инструкцией, как интерпретировать уровни сложности, на что обращать внимание в схеме БД при генерации вопроса и т.д.
3. Запрос выполняется с использованием Structure Output, и ответ парсится в заранее заданном формате через Pydantic-модель `GeneratedQuestion`
4. Сгенерированный запрос валидируется на предмет синтаксических ошибок SQL
5. Сгенерированный запрос выполняется на БД, загруженной для теста
6. Если на проверках в пунктах 4 или 5 оказывается, что запрос не валидный, то генерация повторяется начиная с шага 3.

Листинг 3. Модель для структурированного вывода

```
class GeneratedQuestion(BaseModel):
    thinking: str = Field(
        description="Разбор схемы БД, таблиц и колонок, план SQL-запроса"
    )
    question: str = Field(
        description="Вопрос на русском языке для студента"
    )
    correct_sql: str = Field(
        description="Правильный SQL SELECT-запрос"
    )
```

Класс `ChatOpenAI` с параметром `with_structured_output` гарантирует соблюдение схемы ответа.

Настройка LLM:

Листинг 4. Настройка и запуск цепочки генерации

```
        def _make_llm():
            return ChatOpenAI(
                model=os.getenv("MODEL_NAME", "gpt-4o-mini"),
                base_url=os.getenv("OPENAI_BASE_URL"),
                api_key=os.getenv("OPENAI_API_KEY"),
            )

        def _make_chain():
            llm = _make_llm().with_structured_output(GeneratedQuestion)
            return prompt | llm

        async def generate_question(schema, complexity):
            chain = _make_chain()
            result = await chain.ainvoke({
                "schema": schema,
                "complexity": complexity
            })
            return {
                "question": result.question,
                "correct_sql": result.correct_sql,
                "thinking": result.thinking,
                "time": time.time() - start,
            }
```

Системный промпт включает правила генерации: только SELECT-запросы, шкала сложности 1–5, валидный SQLite-синтаксис. Промпт пользователя содержит схему БД и желаемый уровень сложности.

Благодаря использованию LangChain и совместимого с OpenAI API формата, модель легко заменяется: достаточно изменить переменные окружения `MODEL_NAME`, `OPENAI_BASE_URL` и `OPENAI_API_KEY`. В проекте по умолчанию настроена локальная модель Qwen3:8b через Ollama, работающая по адресу `http://localhost:11434/v1`.

В промышленной среде рекомендуется использовать либо облачных провайдеров, либо производить инференс языковой модели на отдельном сервере с GPU.

4.4.2 Структура базы данных Supabase

Структура хранения данных в Supabase показана на рисунке 4.10.

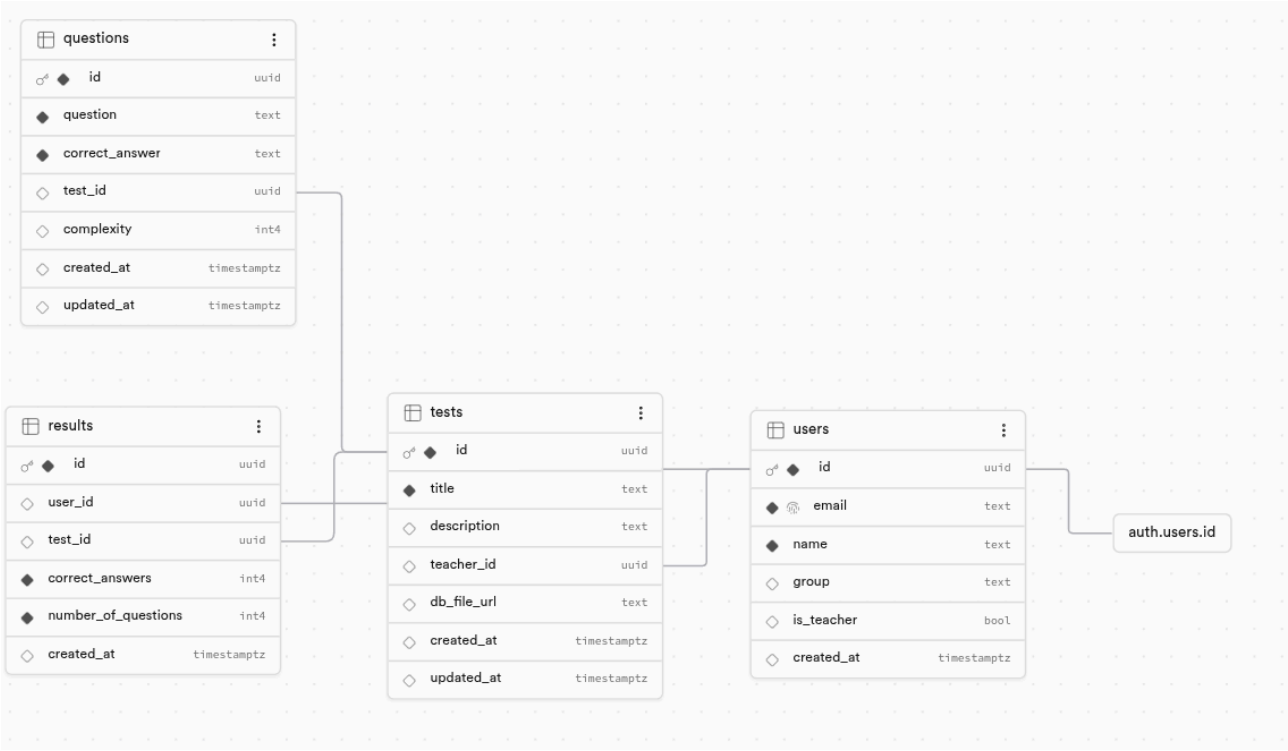

Рисунок 4.10 – Схема базы данных

Сервис использует следующие таблицы в Supabase (PostgreSQL):

Таблица `tests` — метаданные тестов:

- `id` (uuid, PK) — уникальный идентификатор.
- `title` (text) — название теста.
- `description` (text) — описание теста.
- `teacher_id` (uuid, FK -> auth.users.id) — идентификатор преподавателя-создателя.
- `db_file_url` (text) — URL к SQLite-файлу в Storage.
- `created_at` (timestamptz) — дата создания.

Таблица `questions` — вопросы теста:

- `id` (uuid, PK) — идентификатор вопроса.
- `test_id` (uuid, FK -> tests.id) — тест, к которому относится вопрос.
- `question` (text) — текст вопроса на русском языке.
- `correct_answer` (text) — правильный SQL-запрос.
- `complexity` (integer, 1–5) — уровень сложности.
- `created_at` (timestamptz) — дата создания.

Таблица `results` — результаты студентов:

- `id` (uuid, PK).
- `user_id` (uuid, FK -> auth.users.id) — студент.
- `test_id` (uuid, FK -> tests.id) — тест.
- `correct_answers` (integer) — количество правильных ответов.
- `number_of_questions` (integer) — общее количество вопросов.
- `created_at` (timestamptz) — дата и время сдачи.

Таблица `users` — профили пользователей:

- `id` (uuid, PK, FK -> auth.users.id).
- `email` (text) — email пользователя.
- `name` (text) — отображаемое имя.
- `group` (text) — учебная группа (для студентов).
- `is_teacher` (boolean) — флаг роли (true = преподаватель, false = студент).

Хранилище `db-files` предназначено для SQLite-файлов, загруженных преподавателями. Файлы организуются по произвольным именам, а ссылка на файл сохраняется в поле `db_file_url` таблицы `tests`.

4.5 Реализация клиентской части (Frontend)

4.5.1 Маршрутизация и навигация

Приложение использует Vue Router [32] с пятью основными маршрутами:

- `/` — главная страница со списком тестов;
- `/register` — регистрация;
- `/login` — вход;
- `/test/:testId` — страница управления тестом (для преподавателя);
- `/test/solve/:testId` — страница решения теста (для студента).

В работе реализован Navigation-guard — это механизм, который проверяет статус текущего пользователя и запрещает доступ к определенным маршрутам. В частности, проверяет статус аутентификации и перенаправляет неавторизованных пользователей на страницу входа.

4.5.2 Аутентификация и управление состоянием

Аутентификация реализована через Supabase Auth. Vuex [33] store управляет состоянием текущего пользователя. При старте приложения store проверяет наличие сессии в localStorage браузера и при необходимости восстанавливает её.

Сервис `auth.js` (`services/auth.js`) предоставляет функции:

- `getSession()` — получение текущей сессии;
- `signIn(email, password)` — вход;
- `signUp(email, password, userData)` — регистрация;
- `signOut()` — выход;

4.5.3 Компонентная архитектура

Ключевые компоненты Vue:

- `TheHeader.vue` — шапка с логотипом, кнопками входа/регистрации, выпадающим меню пользователя.
- `CreateTestBtn.vue` и `EditTestBtn.vue` — диалоги создания и редактирования теста с загрузкой .db файла в Supabase Storage.
- `TestCard.vue` — карточка теста: отображает название, описание, статус прохождения (для студента), кнопки редактирования/удаления (для преподавателя).

- `QuestionCard.vue` — карточка вопроса с отображением текста, правильного ответа, индикатора сложности, возможностью удаления и редактирования.
- `CreateQuestionBtn.vue` — диалог создания вопроса: ручной ввод или генерация через AI. Поле сложности реализовано через слайдер (v-slider).
- `SqlEditor.vue` — Поле ввода для SQL-кода с поддержкой подсветки синтаксиса и нумерации строк.
- `ResponseTable.vue` — таблица результатов SQL-запроса.
- `ComplexityLine.vue` — 5-сегментный индикатор сложности, который отображается в интерфейсе студента.

4.5.4 Интеграция с бэкендом

Модуль `api/index.js` создаёт два экземпляра axios [34]: `apiClientUserDB` (для работы с сервисом SQLite Query Service) и `apiClientAI` (для работы с сервисом AI Service). Модуль экспортирует два объекта: `user_db_api` и `ai_api`.

Объект `user_db_api` предоставляет методы:

`user_db_api.query_to_user_db(test_id, query)` — используется для выполнения SQL-кода

`user_db_api.create_er_diagram(test_id)` — возвращает ER-диаграмму, закодированную в base64

`user_db_api.get_schema(test_id)` — возвращает текст схемы базы данных

Объект `ai_api` предоставляет метод:

`ai_api.create_question(schema, complexity)` — используется для генерации вопроса на основании схемы базы данных.

4.6 Безопасность

Главный источник риска в этой системе очевиден: студент пишет произвольный SQL-запрос, и сервис его выполняет. Если выполнять такой запрос напрямую на рабочей базе, любой студент сможет удалить чужие данные, прочитать ответы к заданиям или вывести сервис из строя тяжёлым запросом. Поэтому ключевое архитектурное решение здесь — изоляция через временную копию.

Каждый вызов эндпоинта `/sql` не трогает оригинал. Сервис скачивает `.db`-файл из Supabase Storage в отдельный временный файл со случайным именем (UUID), подключается уже к этой копии и удаляет её сразу после выполнения, и при успехе, и при ошибке, что видно по блоку очистки в `finally`-логике функции. Даже если студент отправит `DROP TABLE` или `DELETE FROM`, пострадает только одноразовая копия, которая всё равно будет стёрта через долю секунды. Оригинал в хранилище остаётся нетронутым, а следующий студент получит свежую копию из того же исходного файла.

У такой схемы есть полезный побочный эффект: запросы разных студентов полностью изолированы друг от друга. Никакого общего состояния между сессиями нет, состязаний за доступ к одному файлу не возникает, а параллельные соединения не мешают друг другу, потому что каждое работает со своей копией.

Запросы, которые генерирует AI Service, проходят отдельную проверку. Системный промпт прямо запрещает всё, кроме `SELECT`, а сгенерированный SQL дополнительно валидируется на синтаксис и пробно выполняется на целевой базе: если он не парсится или падает с ошибкой, генерация повторяется. Так эталонные ответы, которые попадают в базу, гарантированно остаются читающими и работоспособными.

Аутентификация и разграничение доступа целиком вынесены в Supabase Auth. После входа клиент получает JWT, который хранится в `localStorage` и прикладывается к запросам к Supabase. На уровне самой PostgreSQL работают политики Row Level Security: преподаватель видит и редактирует только свои тесты, студент — только свои результаты. Это важно, потому что политики применяются в базе, а не в коде фронтенда, и их нельзя обойти, подменив запрос из браузера или обратившись к REST API напрямую.

Разделение ключей тоже сделано осознанно. Фронтенд работает с публичным anon-ключом, у которого прав ровно столько, сколько разрешают RLS-политики. Сервисный ключ с полным доступом к данным лежит только на бэкенде, в переменных окружения сервиса `sqlite_query_service`, и никогда не уходит в браузер. Все секреты (ключи Supabase, адрес и токен LLM) читаются из `.env` и не попадают в репозиторий, для чего в каждом сервисе лежит `.env.example` с описанием нужных переменных без реальных значений.

Доступ к бэкендам со стороны браузера ограничен политикой CORS. Оба FastAPI-сервиса поднимают `CORSMiddleware` и разрешают запросы только с домена фронтенда, так что чужая страница не сможет от имени пользователя обратиться к API напрямую.

5 ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

5.1 Страница аутентификации

Страницы входа и регистрации показаны на рисунках 5.1 и рисунке 5.2 соответственно.

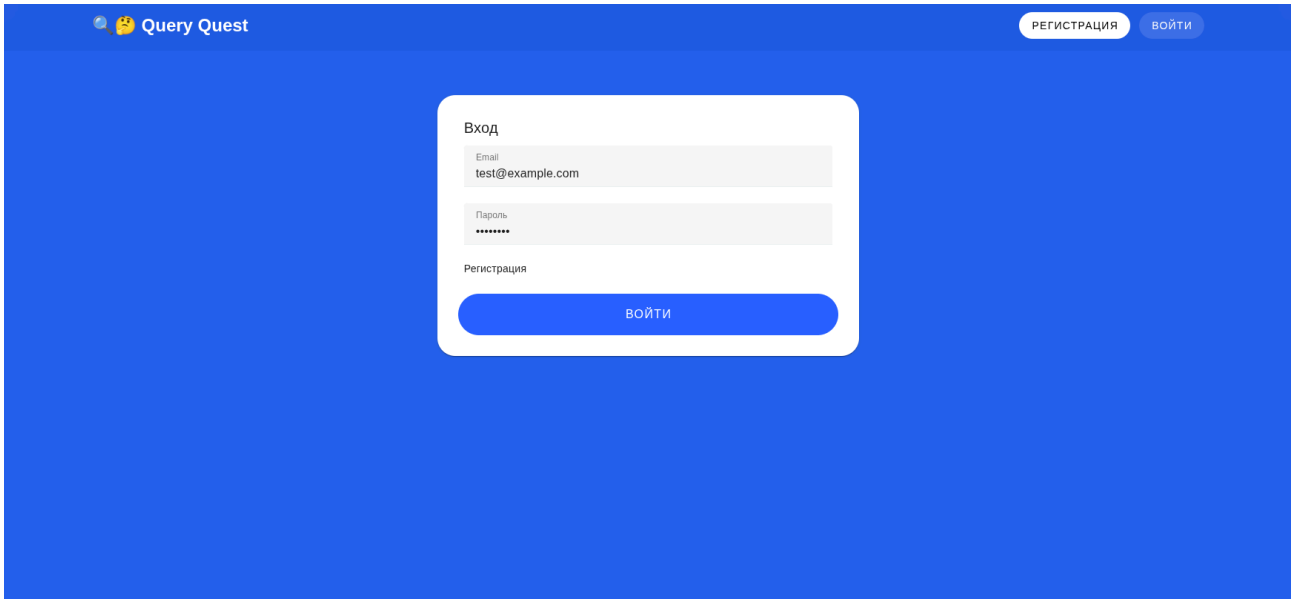

Рисунок 5.1 – Скриншот страницы входа

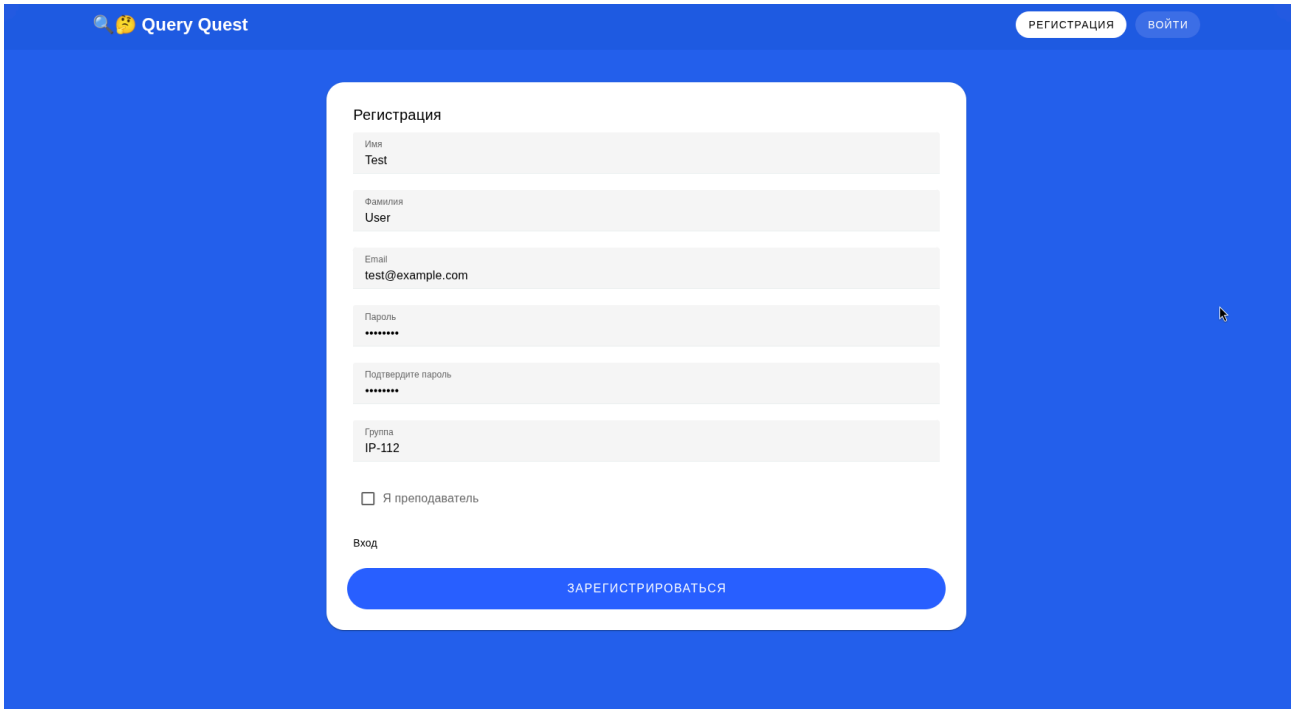

Рисунок 5.2 – Скриншот страницы регистрации

Страницы регистрации и входа реализованы по гайдлайнам дизайн системы Google Material Design 3 с использованием Vuetify-компонентов. Формы

содержат поля email и пароль, а также переключатель роли при регистрации. После успешного входа пользователь перенаправляется на главную страницу.

5.2 Главная страница (список тестов)

Главная страница глазами преподавателя показана на рисунке 5.3.

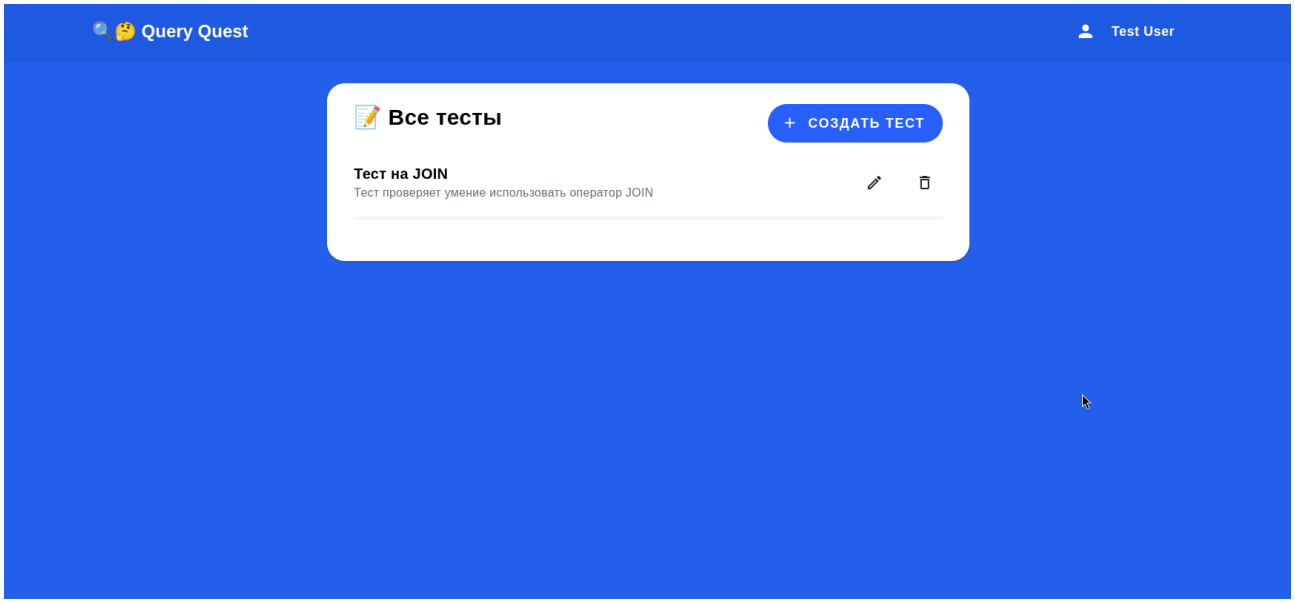

Рисунок 5.3 – Скриншот главной страницы от лица преподавателя

На главной странице отображается список всех доступных тестов. Для преподавателя показываются только его собственные тесты с возможностью редактирования названия, описания и удаления. Для студента отображаются все тесты с индикацией результата прохождения при наличии сохранённого результата.

Кнопка «Создать тест» доступна только преподавателям. Вид главной страницы для студента приведён на рисунке 5.4.

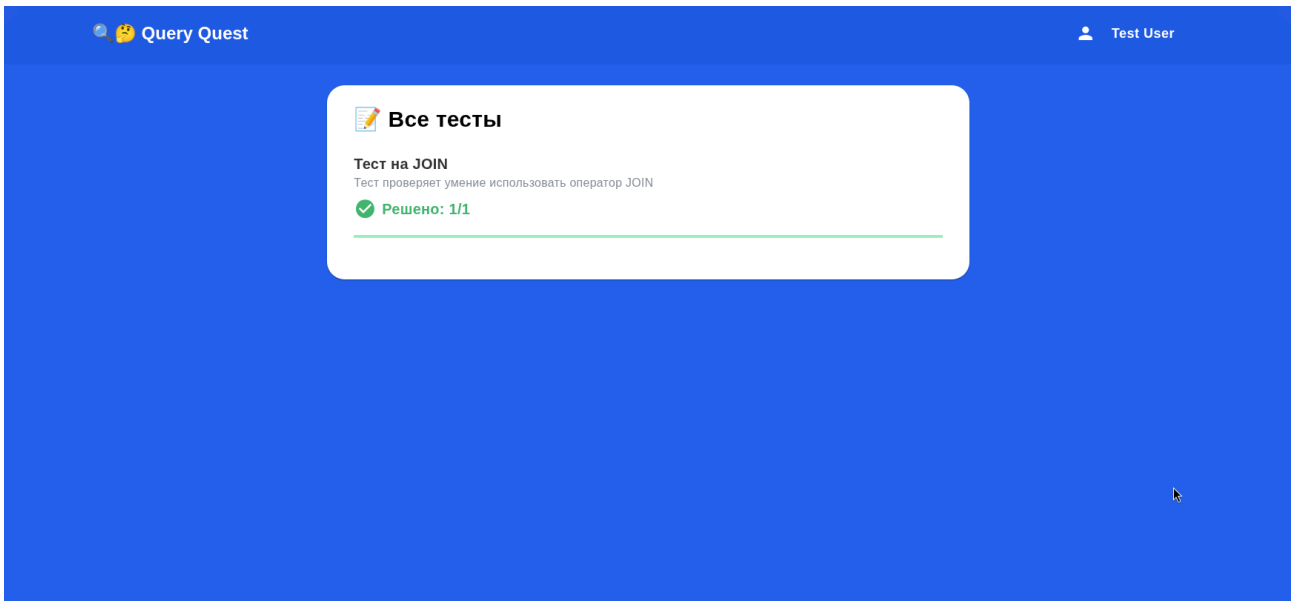

Рисунок 5.4 – Скриншот главной страницы от лица студента

5.3 Страница управления тестом (преподаватель)

Страница управления тестом показана на рисунке 5.5.

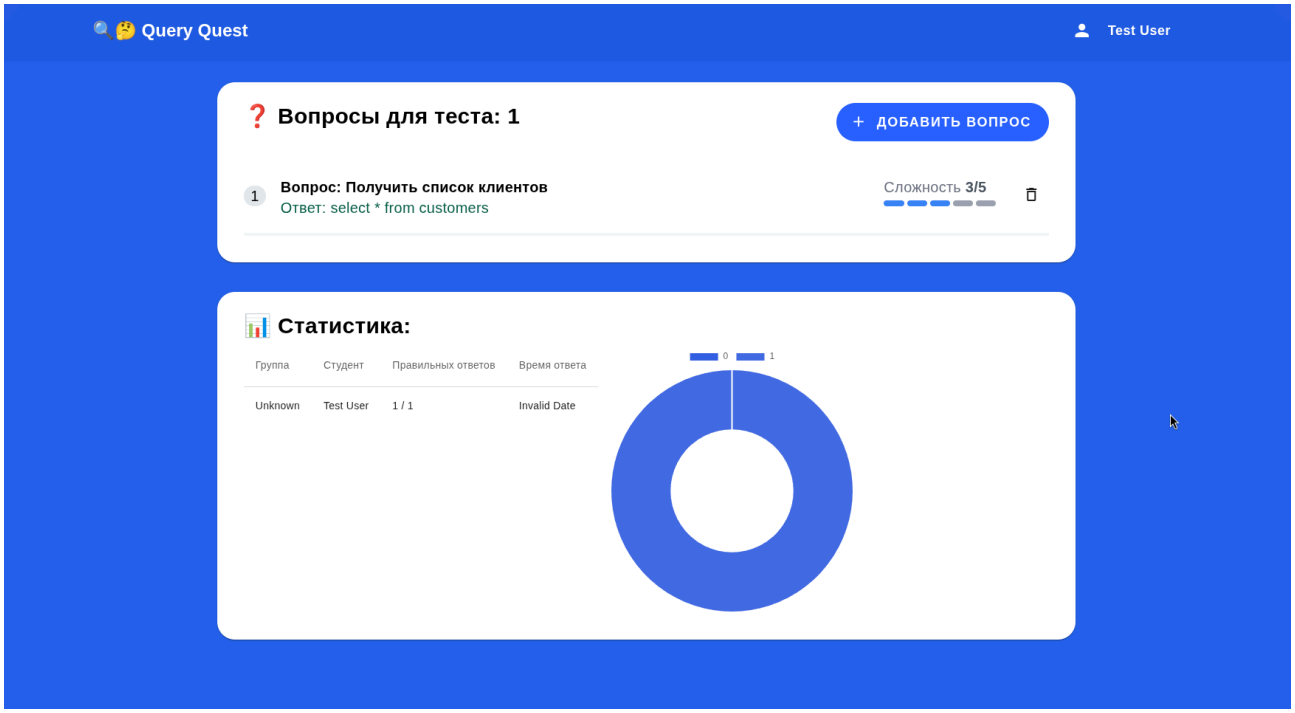

Рисунок 5.5 – Скриншот страницы управления тестом (преподаватель)

В нижней части страницы находится блок статистики, в котором отображаются все прохождения выбранного теста. Для каждого прохождения отображается группа студента, его имя и фамилия, кол-во правильных ответов и время когда был пройден тест.

Справа от таблицы показывается диаграмма pie-chart, которая показывает распределение, сколько студентов решили верно каждое возможное кол-во заданий в тесте.

Страница содержит список вопросов в тесте, для каждого вопроса отображается корректный SQL-ответ и индикатор сложности вопроса. По клику на карточку с вопросом открывается модальное окно с возможностью отредактировать вопрос, ответ и изменить уровень сложности. Справа от индикатора сложности располагается кнопка для удаления вопроса из текущего теста. Модальное окно редактирования вопроса показано на рисунке 5.6.

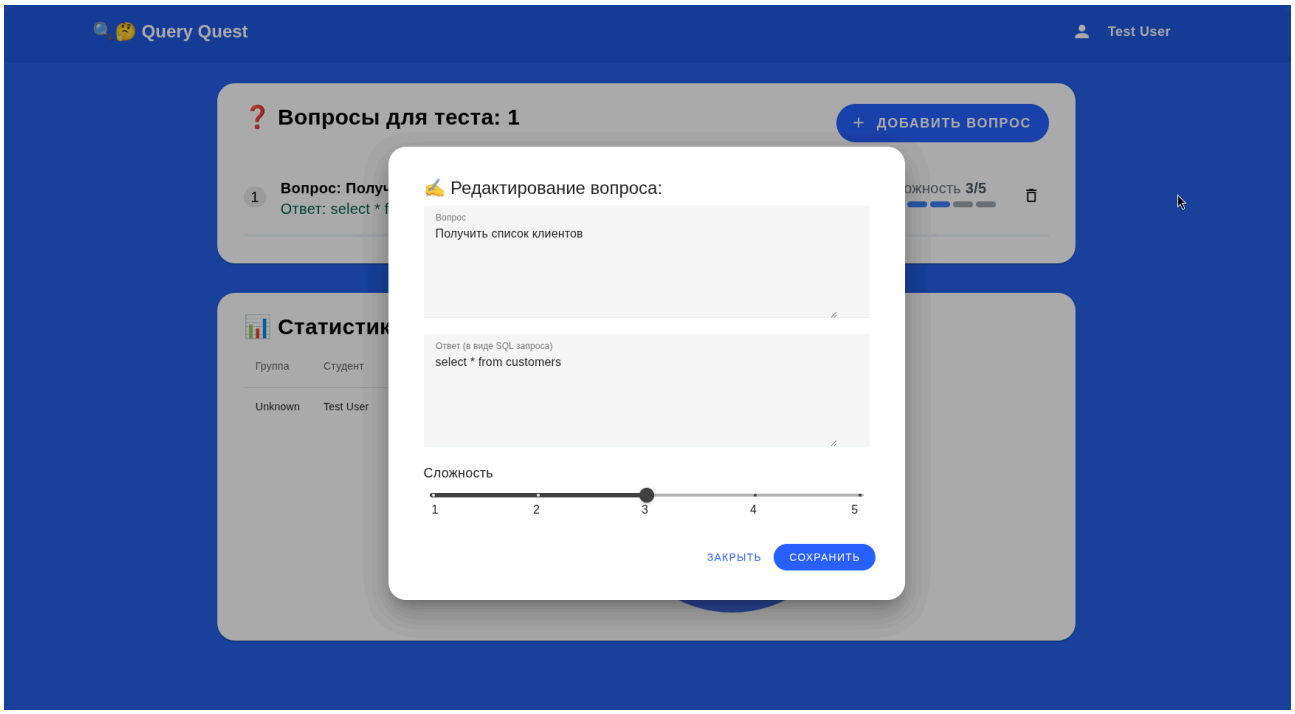

Рисунок 5.6 – Модальное окно редактирования вопроса

Кнопка «Добавить вопрос» открывает модальное окно, в котором преподавателю предлагается вписать формулировку вопроса, указать SQL запрос, который корректно выполняет поставленную в вопросе задачу и указать уровень сложности с помощью слайдера. Модальное окно создания вопроса приведено на рисунке 5.7.

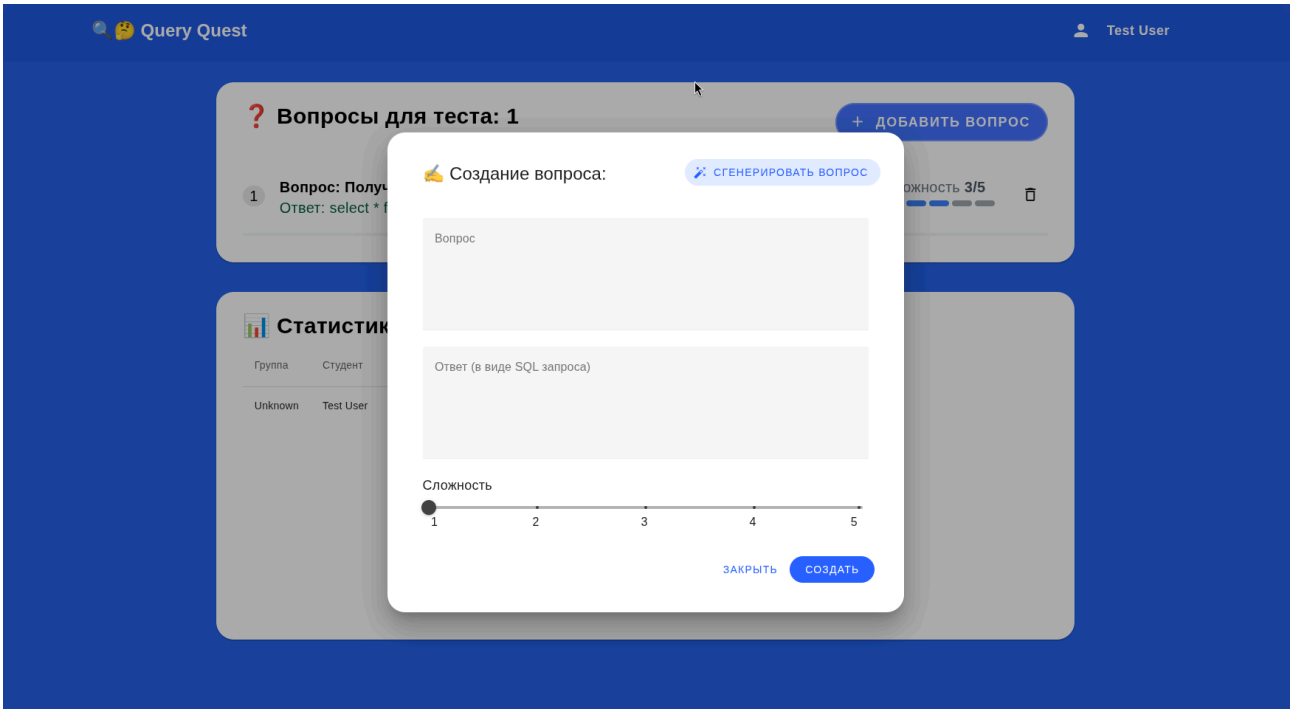

Рисунок 5.7 – Модальное окно создания вопроса

В этом же модальном окне есть кнопка «Сгенерировать вопрос», которая обращается к LLM. Сложность сгенерированного вопроса напрямую зависит от того, какое значение указал пользователь с помощью слайдера.

5.4 Страница решения теста (студент)

Страница решения теста студентом показана на рисунке 5.8.

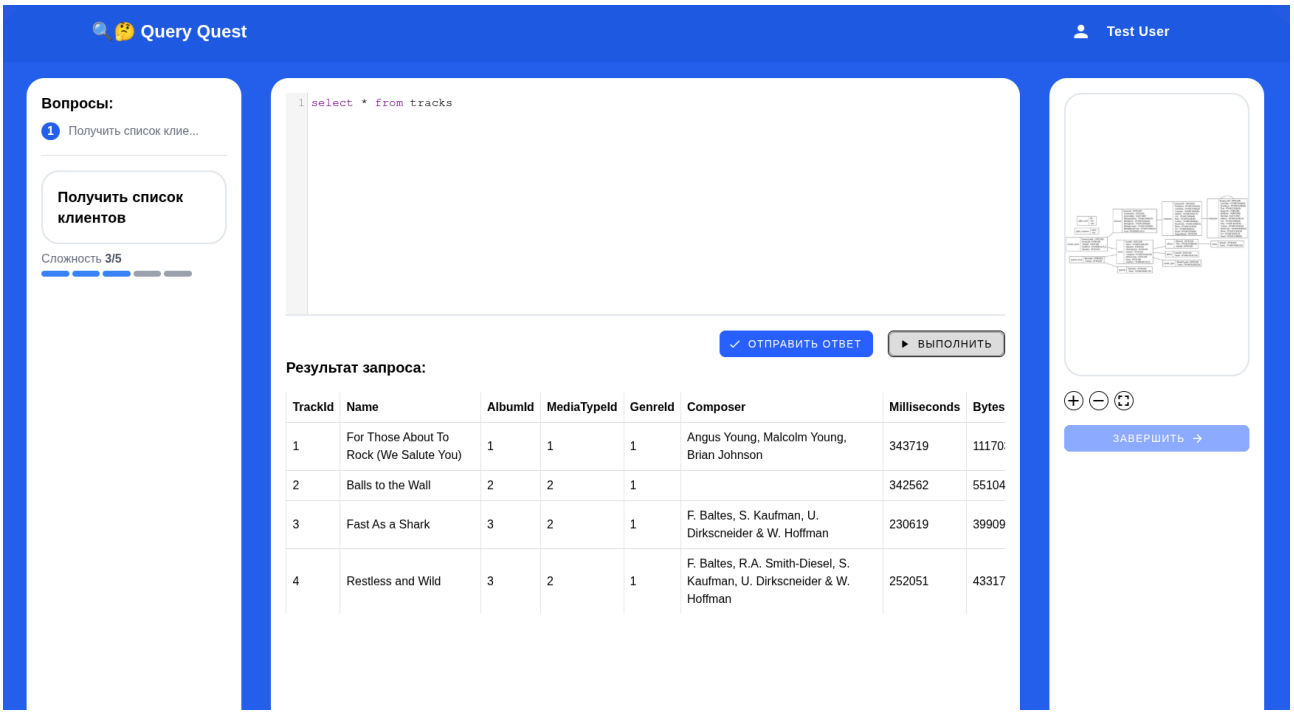

Рисунок 5.8 – Скриншот страницы решения теста от лица студента

Страница решения имеет трёхколоночный макет:

Левая колонка — это список вопросов с номерами. Каждый вопрос отображает статус проверки: зелёная галочка — правильно, красный крестик — неправильно. При нажатии на вопрос в центральной панели отображается его текст и сложность.

Центральная колонка — это редактор SQL-запросов на основе CodeMirror [35] с подсветкой синтаксиса и нумерацией строк. Под редактором расположены:

- Кнопка «Выполнить», которая отправляет запрос на `/sql` сервис, результат отображается в таблице `ResponseTable`.
- Кнопка «Отправить ответ» сравнивает результат студента с эталонным (путём сравнения JSON-представления результирующих наборов данных). Доступна только один раз для каждого вопроса.
- Прогресс-бар при выполнении sql запроса.
- Сообщения об ошибках выполнения SQL.

Правая колонка содержит ER-диаграмму текущей базы данных, отображаемую через компонент `v-zoomer` с возможностью масштабирования и просмотра в полноэкранном режиме. Также этот блок интерфейса содержит кнопку «Завершить», которая становится активной только после отправки ответов на все вопросы.

При завершении результаты сохраняются в таблицу `results`, и студент перенаправляется на главную страницу.

6 ТЕСТИРОВАНИЕ И РАЗВЁРТЫВАНИЕ СИСТЕМЫ

Качество сервиса проверялось с двух сторон. Функциональное тестирование отвечает на вопрос, работают ли пользовательские сценарии целиком, от регистрации до сохранения результата. А качество самой генерации заданий, которое к обычным проверочным сценариям не сводится, измерялось отдельным бенчмарком, описанным в разделе 4: там сравнивались модели и стратегии промптинга по точности калибровки уровней сложности и по оценкам модели-судьи.

6.1 Функциональное тестирование

Тестирование проводилось вручную по сквозным сценариям, покрывающим обе роли и обе серверные части. Каждый сценарий проверял не отдельную функцию в изоляции, а полный путь: действие в интерфейсе, запрос к нужному сервису, ответ и его отображение. Результаты сведены в таблице 6.1.

Таблица 6.1 – Результаты функционального тестирования по сценариям

№	Проверяемый сценарий	Ожидаемый результат	Итог
1	Регистрация с ролью «преподаватель»	Создаётся пользователь, в таблице `users` выставлен флаг `is_teacher`, выполняется переход на главную	Пройден
2	Вход с верными учётными данными	Сессия создаётся, JWT сохраняется в `localStorage`	Пройден
3	Открытие страницы теста без авторизации	Navigation-guard перенаправляет на страницу входа	Пройден
4	Создание теста с загрузкой `.db`-файла	Файл загружается в Storage, в таблице `tests` создаётся запись с `db_file_url`	Пройден
5	Генерация вопроса через ИИ для уровня 3	Возвращается формулировка и валидный `SELECT`, который выполняется на загруженной базе	Пройден
6	Выполнение SQL-запроса студентом	Результат запроса отображается в таблице под редактором	Пройден
7	Проверка верного и неверного ответа	Верный ответ отмечается галочкой, неверный — крестиком; повторная отправка заблокирована	Пройден
8	Завершение теста	Результат записывается в таблицу `results`, происходит переход на главную	Пройден
9	Просмотр ER-диаграммы базы данных	Диаграмма строится и отображается в правой колонке с возможностью масштабирования	Пройден
10	Отправка деструктивного запроса (`DROP TABLE`)	Запрос выполняется на временной копии, исходный файл в Storage остаётся цел	Пройден

Все сценарии отработали как ожидалось. Отдельно проверялись пограничные случаи: пустой результат запроса (запрос корректен, но строк не возвращает), синтаксически неверный SQL (студент должен увидеть текст ошибки, а не падение страницы) и повторный вход уже авторизованного пользователя (сессия восстанавливается из `localStorage` без повторного логина).

6.2 Тестирование безопасности

Главная угроза сервиса — выполнение произвольного SQL от студента, поэтому изоляция временной копии проверялась прицельно. Сценарий 10 из таблицы 6.1 выполнялся явно: на тестовую базу отправлялись `DROP TABLE` и `DELETE FROM`, после чего проверялось, что исходный `.db`-файл в Storage не изменился, а следующий студент получает свежую нетронутую копию. Результат подтвердил ожидание: удаление затронуло только одноразовый временный файл, который сервис всё равно стирает сразу после выполнения запроса.

Проверялось и разграничение доступа. Под учётной записью студента предпринимались попытки получить чужие результаты и отредактировать чужой тест напрямую через REST API Supabase в обход интерфейса: все они отклонялись политиками Row Level Security на стороне базы. Запрос к бэкенду с постороннего origin блокировался политикой CORS.

6.3 Развёртывание системы

Локальная разработка и промышленная эксплуатация устроены по-разному, и удобнее описать их отдельно. На машине разработчика всё запускается через `make dev` в одной tmux-сессии: три процесса (фронтенд, два бэкенда) запускаются рядом, каждый со своим autoreload. Для промышленной эксплуатации этого мало: там нужны воспроизводимая сборка, единая точка входа и HTTPS.

Управление зависимостями. Сервисы на Python используют uv [36] — менеджер пакетов и окружений, написанный на Rust. По сравнению с привычным `pip` он ставит зависимости в разы быстрее и, что важнее для повторяемости сборок, фиксирует точные версии в `uv.lock`. Команда `uv sync` восстанавливает окружение из этого lock-файла бит в бит, поэтому образ, собранный сегодня, ничем не отличается от собранного через месяц. Фронтенд собирается через Bun [37]: он же выступает рантаймом и пакетным менеджером, а `bun run build` собирает проект с помощью Vite и выдаёт статику: набор html, js и css-файлов, которые остаётся раздать веб-сервером.

Контейнеризация. Каждый сервис упакован в отдельный Docker-образ [38], а собираются и связываются они через `docker compose`. Для сервисов на Python используется многоступенчатая (multi-stage) сборка: на первом этапе uv ставит зависимости, на втором в финальный образ копируется только готовое окружение и код приложения, без кеша пакетов и инструментов сборки. Образ получается заметно легче, а слои с зависимостями кешируются между сборками: если поменялся только код, заново загружать пакеты не нужно. Фронтенд собирается так же в два этапа: сначала Bun собирает статику, затем она помещается в лёгкий образ веб-сервера. Compose-файл описывает все сервисы сразу, передаёт переменные окружения из `.env` и создаёт общую сеть, внутри которой контейнеры обращаются друг к другу по именам, а не по IP.

Обратный прокси и HTTPS. Наружу открыт один порт, за которым стоит обратный прокси. Основной вариант — Caddy [39]: он маршрутизирует запросы

(статику фронтенда отдаёт сам, а пути `/api` проксирует на нужный бэкенд) и, что особенно удобно, сам выпускает и продлевает TLS-сертификаты. Caddy из коробки общается с Let’s Encrypt [40] по протоколу ACME, получает бесплатный сертификат для домена и автоматически обновляет его до истечения срока: заниматься сертификатами и перезагрузками вручную не приходится, весь конфиг умещается в несколько строк Caddyfile. Где инфраструктура уже построена вокруг nginx [41], та же схема собирается на нём: nginx раздаёт статику, проксирует API через `proxy_pass` и терминирует TLS, а сертификаты в этом случае выпускаются и продлеваются отдельным клиентом ACME (например, certbot) по расписанию. Caddy здесь предпочтительнее ровно потому, что снимает заботу о сертификатах полностью, тогда как nginx требует настраивать продление вручную.

Прокси решает и вопрос CORS на уровне инфраструктуры: фронтенд и API живут за одним доменом и отличаются только префиксом пути, поэтому для браузера это один источник, и предварительные OPTIONS-запросы отпадают сами собой.

По описанной схеме сервис развёрнут и доступен по адресу `https://queryquest.serveremey.ru`. Обратный прокси Caddy автоматически выпустил для этого домена TLS-сертификат Let’s Encrypt, так что приложение работает по HTTPS, а фронтенд и оба бэкенда обслуживаются за единой точкой входа.

ЗАКЛЮЧЕНИЕ

В результате преддипломной практики был разработан веб-сервис для интерактивного обучения студентов языку SQL.

В ходе выполнения работы были решены следующие задачи:

1. Проведён анализ предметной области и существующих аналогов, выявлены их ограничения в контексте образовательного процесса.
2. Сформулированы функциональные требования к системе и к интерфейсу пользователя.
3. Выбран и обоснован стек технологий.
4. Разработана архитектура системы, включающая два независимых микросервиса.
5. Реализованы все ключевые эндпоинты: выполнение SQL-запросов, генерация ER-диаграмм, получение схемы БД, генерация вопросов через LLM.
6. Разработан пользовательский интерфейс на Vue 3 с поддержкой двух ролей пользователей (преподаватель, студент).
7. Реализована интеграция с LLM, обеспечивающая автоматическую генерацию вопросов на основе анализа схемы любой загруженной базы данных.
8. Проведено функциональное тестирование сквозных сценариев и проверка изоляции выполнения SQL, система подготовлена к контейнерному развёртыванию.

В результате система готова к использованию в учебном процессе. Автоматическая генерация вопросов позволяет преподавателям создавать разнообразные тестовые задания, существенно сокращая время на подготовку материалов. Визуализация схем данных в виде ER-диаграмм помогает студентам быстрее понимать структуру базы данных. Механизм автоматической проверки через сравнение результирующих наборов данных обеспечивает объективную оценку навыков студента.

Перспективы дальнейшего развития включают: расширение типов генерируемых заданий (INSERT, UPDATE, DELETE), добавление групповой аналитики для потоков студентов, интеграцию с системами управления обучением.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. U.S. Bureau of Labor Statistics. Occupational Outlook Handbook: Data Scientists. Таблица 6: Employment change and projected growth [Электронный ресурс]. 2025. URL: https://www.bls.gov/ooh/math/data-scientists.htm#tab-6 (дата обращения: 14.05.2025).
2. Stack Overflow. Developer Survey 2023 [Электронный ресурс]. 2023. URL: https://survey.stackoverflow.co/2023 (дата обращения: 14.05.2025).
3. Verougstraete R. What Job Postings Say About Demand for SQL [Электронный ресурс]. Lightcast, 2021. URL: https://lightcast.io/resources/blog/what-job-postings-say-about-demand-for-sql (дата обращения: 14.05.2025).
4. Мейкшан В. И. Основы языка SQL в примерах и задачах. Новосибирск: СибГУТИ, 2013. С. 41.
5. Taipalus T., Seppänen V. SQL and database education: обзор рекомендаций по практическому обучению и будущая повестка исследований [Электронный ресурс]. ResearchGate, 2020. URL: https://www.researchgate.net/publication/342759889_SQL_Education_A_Systematic_Mapping_Study_and_Future_Research_Agenda (дата обращения: 14.05.2025).
6. Vaswani A. и др. Attention Is All You Need // Advances in Neural Information Processing Systems. 2017. Т. 30.
7. Frantar E. и др. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers // arXiv preprint arXiv:2210.17323. 2022.
8. Lin J. и др. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration // arXiv preprint arXiv:2306.00978. 2023.
9. Gerganov G. llama.cpp — LLM inference in C/C++ [Электронный ресурс]. 2025. URL: https://github.com/ggml-org/llama.cpp (дата обращения: 10.06.2025).
10. Hiltgen D. ollama/docs [Электронный ресурс]. 2025. URL: https://github.com/ollama/ollama/tree/main/docs (дата обращения: 14.05.2025).
11. Supabase. Supabase Documentation [Электронный ресурс]. 2025. URL: https://supabase.com/docs (дата обращения: 20.05.2025).
12. Ramirez S. FastAPI — документация [Электронный ресурс]. 2025. URL: https://fastapi.tiangolo.com/ (дата обращения: 20.05.2025).
13. Encode. Starlette — документация [Электронный ресурс]. 2025. URL: https://www.starlette.io/ (дата обращения: 20.05.2025).
14. Pydantic. Pydantic — документация [Электронный ресурс]. 2025. URL: https://docs.pydantic.dev/latest/ (дата обращения: 20.05.2025).
15. Encode. Uvicorn — документация [Электронный ресурс]. 2025. URL: https://www.uvicorn.org/ (дата обращения: 20.05.2025).
16. Vue.js. Vue.js 3 — официальное руководство [Электронный ресурс]. 2025. URL: https://vuejs.org/guide/introduction.html (дата обращения: 20.05.2025).
17. OpenAI. OpenAI API Reference [Электронный ресурс]. 2025. URL: https://platform.openai.com/docs/api-reference (дата обращения: 20.05.2025).

18. Team Q. Qwen 3: Think Deeper, Act Faster [Электронный ресурс]. 2025. URL: https://qwenlm.github.io/blog/qwen3/ (дата обращения: 14.05.2025).
19. LangChain. LangChain — документация [Электронный ресурс]. 2025. URL: https://python.langchain.com/docs/introduction/ (дата обращения: 20.05.2025).
20. Mohammadjafari A., Maida A. S., Gottumukkala R. From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems // arXiv preprint arXiv:2410.01066. 2024.
21. Dong X. и др. C3: Zero-shot Text-to-SQL with ChatGPT // arXiv preprint arXiv:2307.07306. 2023.
22. Zheng L. и др. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena // Advances in Neural Information Processing Systems. 2023. T. 36.
23. Liu Y. и др. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment // Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2023.
24. Wang P. и др. Large Language Models are not Fair Evaluators // arXiv preprint arXiv:2305.17926. 2023.
25. Panickssery A., Bowman S. R., Feng S. LLM Evaluators Recognize and Favor Their Own Generations // Advances in Neural Information Processing Systems. 2024. T. 37.
26. Spiliopoulou E. и др. Play Favorites: A Statistical Method to Measure Self-Bias in LLM-as-a-Judge // arXiv preprint arXiv:2508.06709. 2025.
27. Tailwind Labs. Tailwind CSS — документация [Электронный ресурс]. 2025. URL: https://tailwindcss.com/docs/installation (дата обращения: 20.05.2025).
28. Vuetify. Vuetify 3 — официальная документация [Электронный ресурс]. 2025. URL: https://vuetifyjs.com/en/getting-started/installation/ (дата обращения: 20.05.2025).
29. Google. Material Design 3 [Электронный ресурс]. 2025. URL: https://m3.material.io/ (дата обращения: 20.05.2025).
30. Python Software Foundation. sqlite3 — DB-API 2.0 interface for SQLite databases [Электронный ресурс]. 2025. URL: https://docs.python.org/3/library/sqlite3.html (дата обращения: 20.05.2025).
31. Carrera E. pydot — Python interface to Graphviz [Электронный ресурс]. 2025. URL: https://github.com/pydot/pydot (дата обращения: 20.05.2025).
32. Vue.js. Vue Router — официальное руководство [Электронный ресурс]. 2025. URL: https://router.vuejs.org/guide/ (дата обращения: 20.05.2025).
33. Vue.js. Vuex — официальное руководство [Электронный ресурс]. 2025. URL: https://vuex.vuejs.org/guide/ (дата обращения: 20.05.2025).
34. Axios. Axios — документация [Электронный ресурс]. 2025. URL: https://axios-http.com/docs/intro (дата обращения: 20.05.2025).
35. CodeMirror. CodeMirror 6 — документация [Электронный ресурс]. 2025. URL: https://codemirror.net/docs/ (дата обращения: 20.05.2025).

36. Astral. uv — An extremely fast Python package and project manager [Электронный ресурс]. 2025. URL: https://docs.astral.sh/uv/ (дата обращения: 10.06.2025).
37. Oven. Bun — A fast all-in-one JavaScript runtime [Электронный ресурс]. 2025. URL: https://bun.sh/docs (дата обращения: 10.06.2025).
38. Docker, Inc. Docker Documentation [Электронный ресурс]. 2025. URL: https://docs.docker.com/ (дата обращения: 10.06.2025).
39. Light Code Labs. Caddy — The Ultimate Server with Automatic HTTPS [Электронный ресурс]. 2025. URL: https://caddyserver.com/docs/ (дата обращения: 10.06.2025).
40. Internet Security Research Group. Let's Encrypt — Free SSL/TLS Certificates [Электронный ресурс]. 2025. URL: https://letsencrypt.org/docs/ (дата обращения: 10.06.2025).
41. F5, Inc. nginx documentation [Электронный ресурс]. 2025. URL: https://nginx.org/en/docs/ (дата обращения: 10.06.2025).

ПРИЛОЖЕНИЕ А. Исходный код сервиса генерации вопросов

Модуль `main.py` — точка входа сервиса:

```
from dotenv import load_dotenv

load_dotenv()

from fastapi import FastAPI
import uvicorn
from fastapi.middleware.cors import CORSMiddleware
from endpoints import create_question

app = FastAPI(title="AI Question Generator")

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(create_question.router)

if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=8081, reload=True)
```

Модуль `utils/questions_generator.py` — цепочка генерации вопроса через LLM:

```
import os
import time

from pydantic import BaseModel, Field
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate


class GeneratedQuestion(BaseModel):
    thinking: str = Field(description="Разбор схемы БД, таблиц и колонок,
план SQL-запроса")
    question: str = Field(description="Вопрос на русском языке для
студента")
    correct_sql: str = Field(description="Правильный SQL SELECT-запрос к
этому вопросу")


def _make_llm():
    return ChatOpenAI(
        model=os.getenv("MODEL_NAME", "gpt-4o-mini"),
        base_url=os.getenv("OPENAI_BASE_URL"),
        api_key=os.getenv("OPENAI_API_KEY"),
    )


SYSTEM_PROMPT = """You are an AI that generates SQL test questions for
students.
```

```
For each request you must follow this exact process:
1. Think carefully about the schema, tables, columns, and relationships.
2. Formulate a question that requires writing a SELECT query. The question
MUST be in Russian.
3. Write the correct SQL SELECT query as the answer.

Rules:
- Generate ONLY questions answerable with a SELECT query (no INSERT/UPDATE/
DELETE).
- Difficulty scale 1-5: 1 = trivial SELECT *, 2 = WHERE filter, 3 = JOIN or
aggregate, 4 = JOIN + GROUP BY + filter, 5 = subqueries, multiple JOINs,
complex logic.
- The question must be clear and answerable based on the given schema.
- The SQL answer must be valid SQLite syntax.
- Think in English internally, but write the question in Russian."""

HUMAN_TEMPLATE = """Database schema:
{schema}

Generate a difficulty {complexity} SQL question. Follow the process: think,
write question (RU), write correct SQL."""

prompt = ChatPromptTemplate.from_messages([
    ("system", SYSTEM_PROMPT),
    ("human", HUMAN_TEMPLATE),
])


def _make_chain():
    llm = _make_llm().with_structured_output(GeneratedQuestion)
    return prompt | llm


async def generate_question(schema: str, complexity: str) -> dict:
    start = time.time()
    safe_schema = schema[:3000]
    chain = _make_chain()
    result: GeneratedQuestion = await chain.ainvoke({"schema": safe_schema,
"complexity": complexity})
    return {
        "question": result.question,
        "correct_sql": result.correct_sql,
        "thinking": result.thinking,
        "time": time.time() - start,
    }
```

Модуль `endpoints/create_question.py` — эндпоинт генерации вопроса:

```
from fastapi import APIRouter, HTTPException
from utils.questions_generator import generate_question

router = APIRouter()


@router.get("/create_question")
async def create_question(schema: str, complexity: str):
    try:
```

```
        return await generate_question(schema, complexity)
    except HTTPException:
        raise
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

ПРИЛОЖЕНИЕ Б. Исходный код сервиса выполнения SQL-запросов

Модуль `main.py` — регистрация эндпоинтов сервиса:

```
from fastapi import FastAPI
import uvicorn
from endpoints import sql, create_er_diagram, get_schema
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(sql.router)
app.include_router(create_er_diagram.router)
app.include_router(get_schema.router)

if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=8080, reload=True)
```

Модуль `supabase_client.py` — работа с REST API и хранилищем Supabase:

```
import os
from typing import Optional
import requests
from pathlib import Path

env_path = Path(__file__).parent / ".env"
if env_path.exists():
    from dotenv import load_dotenv

    load_dotenv(env_path)

SUPABASE_URL = os.getenv("SUPABASE_URL")
SUPABASE_SERVICE_KEY = os.getenv("SUPABASE_SERVICE_KEY")

SUPABASE_REST_URL = f"{SUPABASE_URL}/rest/v1"
SUPABASE_STORAGE_URL = f"{SUPABASE_URL}/storage/v1"


def get_headers():
    return {
        "apikey": SUPABASE_SERVICE_KEY,
        "Authorization": f"Bearer {SUPABASE_SERVICE_KEY}",
        "Content-Type": "application/json",
        "Accept": "application/json",
    }


def get_test_record(record_id: str) -> Optional[dict]:
    url = f"{SUPABASE_REST_URL}/tests"
    params = {"id": f"eq.{record_id}"}
    try:
```

```
        response = requests.get(url, headers=get_headers(), params=params)
        response.raise_for_status()
        records = response.json()
        if records and len(records) > 0:
            return records[0]
        return None
    except requests.exceptions.RequestException as e:
        raise RuntimeError(f"Failed to fetch test record: {e}")


def get_db_file_url(record_id: str, db_filename: str) -> str:
    url = f"{SUPABASE_STORAGE_URL}/object/sign/db-files/{record_id}/
{db_filename}"
    try:
        response = requests.post(
            url,
            headers=get_headers(),
            json={"expiresIn": 3600},
        )
        response.raise_for_status()
        data = response.json()
        signed_url = data.get("signedUrl")
        if signed_url:
            return signed_url
        return f"{SUPABASE_URL}/storage/v1/object/public/db-files/
{record_id}/{db_filename}"
    except requests.exceptions.RequestException:
        return f"{SUPABASE_URL}/storage/v1/object/public/db-files/
{record_id}/{db_filename}"


def download_db_file(url: str, destination: str) -> None:
    try:
        response = requests.get(url, headers=get_headers(), stream=True)
        response.raise_for_status()
        with open(destination, "wb") as f:
            for chunk in response.iter_content(chunk_size=8192):
                f.write(chunk)
    except requests.exceptions.RequestException as e:
        raise RuntimeError(f"Failed to download database file: {e}")
```

Модуль `endpoints/sql.py` — выполнение SQL-запроса на временной копии базы:

```
from fastapi import APIRouter, HTTPException
import sqlite3
import uuid
import os
from supabase_client import get_test_record, download_db_file

router = APIRouter()


@router.get("/sql")
async def execute_sql(record_id: str, query: str):
    file_name = None
    conn = None
    try:
```

```
        record = get_test_record(record_id)
        if not record:
            raise HTTPException(status_code=404, detail=f"Record with ID
{record_id} not found")
        db_file_url = record.get("db_file_url")
        if not db_file_url:
            raise HTTPException(status_code=400, detail="Record does not
contain a database file")
        file_name = str(uuid.uuid4())
        download_db_file(db_file_url, file_name)
        conn = sqlite3.connect(file_name)
        cursor = conn.cursor()
        cursor.execute(query)
        result_data = cursor.fetchall()
        column_names = [desc[0] for desc in cursor.description]
        result = [
            {column: value for column, value in zip(column_names, row)}
            for row in result_data
        ]
        conn.close()
        os.remove(file_name)
        return {"columns": column_names, "result": result}
    except HTTPException:
        raise
    except Exception as e:
        if conn is not None:
            conn.close()
        if file_name and os.path.exists(file_name):
            os.remove(file_name)
        raise HTTPException(status_code=500, detail=str(e))
```

Модуль `endpoints/get_schema.py` — извлечение текстовой схемы базы данных:

```
from fastapi import APIRouter, HTTPException
import sqlite3
import uuid
import os
import re
from supabase_client import get_test_record, download_db_file

router = APIRouter()


@router.get("/schema")
async def get_schema(record_id: str):
    file_name = None
    conn = None
    try:
        record = get_test_record(record_id)
        if not record:
            raise HTTPException(status_code=404, detail=f"Record with ID
{record_id} not found")
        db_file_url = record.get("db_file_url")
        if not db_file_url:
            raise HTTPException(status_code=400, detail="Record does not
contain a database file")
        file_name = str(uuid.uuid4())
```

```
        download_db_file(db_file_url, file_name)
        conn = sqlite3.connect(file_name)
        schema_query = """
            SELECT GROUP_CONCAT(tbl.sql, '\n') AS schema
            FROM sqlite_master AS tbl
            WHERE type='table' AND name NOT LIKE 'sqlite_%';
        """
        cursor = conn.cursor()
        cursor.execute(schema_query)
        pattern = re.compile(r'\s+[A-Z]+\(?[0-9]*\)?\s+')
        schema_line = cursor.fetchone()[0]
        final = pattern.sub(' ', schema_line).replace(" NOT NULL,", "")
        conn.close()
        os.remove(file_name)
        return {"schema": final}
    except HTTPException:
        raise
    except Exception as e:
        if conn is not None:
            conn.close()
        if file_name and os.path.exists(file_name):
            os.remove(file_name)
        raise HTTPException(status_code=500, detail=str(e))
```

Модуль `endpoints/create_er_diagram.py` — эндпоинт генерации ER-диаграммы:

```
import base64
import os
import uuid
from fastapi import APIRouter, HTTPException
from supabase_client import get_test_record, download_db_file
from utils.diagram_creator import generate_er_diagram

router = APIRouter()


@router.get("/er-diagram")
async def generate_er_diagram_endpoint(record_id: str):
    file_name = None
    try:
        record = get_test_record(record_id)
        if not record:
            raise HTTPException(status_code=404, detail=f"Record with ID
{record_id} not found")
        db_file_url = record.get("db_file_url")
        if not db_file_url:
            raise HTTPException(status_code=400, detail="Record does not
contain a database file")
        file_name = str(uuid.uuid4())
        download_db_file(db_file_url, file_name)
        er_diagram_binary = generate_er_diagram(file_name)
        os.remove(file_name)
        er_diagram_base64 =
base64.b64encode(er_diagram_binary).decode("utf-8")
        return {"er_diagram": er_diagram_base64}
    except HTTPException:
        raise
```

```
    except Exception as e:
        if file_name and os.path.exists(file_name):
            os.remove(file_name)
        raise HTTPException(status_code=500, detail=str(e))
```

Модуль `utils/diagram_creator.py` — построение ER-диаграммы по метаданным SQLite:

```
import sqlite3
import pydot


def generate_er_diagram(database_file):
    conn = sqlite3.connect(database_file)
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
    tables = cursor.fetchall()
    table_info = {}
    for table in tables:
        cursor.execute(f"PRAGMA table_info({table[0]});")
        columns = cursor.fetchall()
        cursor.execute(f"PRAGMA foreign_key_list({table[0]});")
        foreign_keys = cursor.fetchall()
        table_info[table[0]] = (columns, foreign_keys)
    graph = pydot.Dot(graph_type='graph', rankdir='LR')
    for table, (columns, foreign_keys) in table_info.items():
        node = pydot.Node(table, shape='record')
        label = f'{{ {table} | {{'
        for column in columns:
            column_name = column[1]
            label += f' {column_name} : {column[2]}\\l'
        label += '} }}'
        node.set_label(label)
        graph.add_node(node)
        for foreign_key in foreign_keys:
            source_table = table
            source_column = foreign_key[3]
            target_table = foreign_key[2]
            target_column = foreign_key[4]
            if source_table < target_table:
                arrowhead = 'crow'
            else:
                arrowhead = 'tee'
            edge = pydot.Edge(f"{source_table}:{source_column}",
f"{target_table}:{target_column}", arrowhead=arrowhead)
            graph.add_edge(edge)
    return graph.create(format='png')
```

ПРИЛОЖЕНИЕ В. Исходный код клиентской части

Модуль `supabase.js` — инициализация клиента Supabase:

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL
const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY

export const supabase = createClient(supabaseUrl, supabaseAnonKey, {
    auth: {
        autoRefreshToken: true,
        persistSession: true,
        detectSessionInUrl: true
    }
})

export default supabase
```

Модуль `api/index.js` — клиенты для обращения к микросервисам:

```
import axios from 'axios'
import { API_URL_USER_DB, API_URL_AI } from './config'

const apiClientUserDB = axios.create({
  baseURL: API_URL_USER_DB,
  'accept': 'application/json',
})

const apiClientAI = axios.create({
  baseURL: API_URL_AI,
  'accept': 'application/json',
})

const user_db_api = {
  query_to_user_db: (test_id, query) => {
    return apiClientUserDB.get(`/sql?record_id=${test_id}&query=${query}`);
  },
  create_er_diagram: (test_id) => {
    return apiClientUserDB.get(`/er-diagram?record_id=${test_id}`);
  },
  get_schema: (test_id) => {
    return apiClientUserDB.get(`/schema?record_id=${test_id}`);
  },
};

const ai_api = {
  create_question: (schema, complexity) => {
    return apiClientAI.get(`/create_question?schema=${schema}
&complexity=${complexity}`);
  }
}

export { user_db_api, ai_api };
```

Модуль `services/auth.js` — сервис аутентификации поверх Supabase Auth:

```
import { supabase } from '../supabase.js';

let authReadyPromise = null;
let isAuthReady = false;

export async function getSession() {
  const { data: { session } } = await supabase.auth.getSession();
  return session;
}

export async function getUserProfile(userId) {
  if (!userId) return null;
  const { data: profile, error } = await supabase
    .from('users')
    .select('id, email, name, "group", is_teacher')
    .eq('id', userId)
    .single();
  if (error) {
    return null;
  }
  return profile;
}

export async function enrichUserWithProfile(user) {
  if (!user) return null;
  const profile = await getUserProfile(user.id);
  return {
    ...user,
    ...profile,
    name: profile?.name || user.user_metadata?.name || user.email,
    is_teacher: profile?.is_teacher ?? user.user_metadata?.is_teacher ??
false,
    group: profile?.group || user.user_metadata?.group || ''
  };
}

export async function getCurrentUser() {
  const session = await getSession();
  if (!session) return null;
  return await enrichUserWithProfile(session.user);
}

export async function signIn(email, password) {
  const { data, error } = await supabase.auth.signInWithPassword({ email,
password });
  if (error) throw error;
  return data;
}

export async function signUp(email, password, userData = {}) {
  const { data, error } = await supabase.auth.signUp({
    email,
    password,
    options: { data: userData }
  });
  if (error) throw error;
  return data;
}
```

```
export async function signOut() {
  const { error } = await supabase.auth.signOut();
  if (error) throw error;
}

export function onAuthStateChange(callback) {
  return supabase.auth.onAuthStateChange(async (event, session) => {
    callback(event, session);
  });
}

export default {
  getSession,
  getCurrentUser,
  getUserProfile,
  enrichUserWithProfile,
  signIn,
  signUp,
  signOut,
  onAuthStateChange
};
```

Модуль `store.js` — управление состоянием пользователя через Vuex:

```
import { createStore } from 'vuex';
import authService from './services/auth.js';

const store = createStore({
  state: {
    currentUser: null,
    isInitialized: false,
  },
  mutations: {
    setCurrentUser(state, user) {
      state.currentUser = user;
    },
    setInitialized(state, value) {
      state.isInitialized = value;
    },
  },
  actions: {
    async initializeAuth({ commit }) {
      try {
        const session = await authService.getSession();
        if (session) {
          const user = await
authService.enrichUserWithProfile(session.user);
          commit('setCurrentUser', user);
        } else {
          commit('setCurrentUser', null);
        }
      } catch (error) {
        commit('setCurrentUser', null);
      }
      commit('setInitialized', true);
    },
    async handleAuthChange({ commit }, session) {
      if (session) {
```

```
        const user = await authService.enrichUserWithProfile(session.user);
        commit('setCurrentUser', user);
      } else {
        commit('setCurrentUser', null);
      }
    },
  },
  getters: {
    currentUser: state => state.currentUser,
    isLoggedIn: state => !!state.currentUser,
    isInitialized: state => state.isInitialized,
  },
});

export default store;
```

Модуль `router.js` — маршрутизация и navigation-guard:

```
import { createRouter, createWebHashHistory } from 'vue-router';
import store from './store';

import RegisterPage from './pages/RegistrationPage.vue';
import HomePage from './pages/HomePage.vue';
import LoginPage from './pages/LoginPage.vue';
import TestPage from './pages/TestPage.vue';
import SolveTestPage from './pages/SolveTestPage.vue';

const router = createRouter({
  history: createWebHashHistory(),
  routes: [
    { path: '/', component: HomePage },
    { path: '/register', component: RegisterPage },
    { path: '/login', component: LoginPage },
    { path: '/test/:testId', component: TestPage, props: true },
    { path: '/test/solve/:testId', component: SolveTestPage, props: true },
    { path: '/:pathMatch(.*)*', redirect: '/' },
  ]
});

router.beforeEach((to, from, next) => {
  const publicPages = ['/login', '/register'];
  const isPublicPage = publicPages.some(path => to.path.startsWith(path));
  const isLoggedIn = store.getters.isLoggedIn;
  const isInitialized = store.getters.isInitialized;
  if (!isInitialized) {
    next();
    return;
  }
  if (isPublicPage && isLoggedIn) {
    next('/');
    return;
  }
  if (!isPublicPage && !isLoggedIn) {
    next('/login');
    return;
  }
  next();
});
```

```
export default router;
```

Компонент `CreateTestBtn.vue` — создание теста с загрузкой `.db`-файла в Storage (логика):

```
import supabase from "../supabase.js";
import { mapGetters } from 'vuex'
import { emitter } from '../eventBus'

export default {
    data() {
        return {
            dialog: false,
            title: '',
            description: '',
            selectedFile: null,
            errorMessage: '',
        }
    },
    computed: {
        ...mapGetters(['currentUser']),
    },
    methods: {
        async create_test() {
            if (!this.title?.trim()) {
                this.errorMessage = 'Укажите название теста';
                return;
            }
            let dbFileUrl = null;
            const normalizedFile = Array.isArray(this.selectedFile)
                ? this.selectedFile[0]
                : this.selectedFile;
            if (normalizedFile && normalizedFile instanceof File) {
                const fileName = `${Date.now()}_${normalizedFile.name}`;
                const { data, error: uploadError } = await supabase.storage
                    .from('db-files')
                    .upload(fileName, normalizedFile);
                if (uploadError) {
                    this.errorMessage = 'Ошибка загрузки файла: ' +
uploadError.message;
                    return;
                }
                const { data: urlData } = await supabase.storage
                    .from('db-files')
                    .getPublicUrl(fileName);
                dbFileUrl = urlData.publicUrl;
            }
            try {
                const { error } = await supabase
                    .from('tests')
                    .insert({
                        title: this.title,
                        description: this.description,
                        teacher_id: this.currentUser.id,
                        db_file_url: dbFileUrl
                    });
                if (error) throw error;
```

```
                emitter.emit('testsUpdated');
                this.dialog = false;
                this.title = '';
                this.description = '';
                this.selectedFile = null;
                this.errorMessage = '';
            } catch (error) {
                this.errorMessage = error.message;
            }
        }
    },
}
```

Компонент `CreateQuestionBtn.vue` — ручное создание и генерация вопроса через ИИ (логика):

```
import supabase from "../supabase.js";
import { mapGetters } from 'vuex'
import { emitter } from '../eventBus'
import { ai_api, user_db_api } from "../api/index"

export default {
    data() {
        return {
            dialog: false,
            question: '',
            answer: '',
            complexity: 1,
            errorMessage: '',
            successMessage: '',
            tickLabels: { 1: '1', 2: '2', 3: '3', 4: '4', 5: '5' },
            loading: false
        }
    },
    computed: {
        ...mapGetters(['currentUser']),
    },
    props: ['test_id'],
    methods: {
        async create_question() {
            if (!this.question?.trim()) {
                this.errorMessage = 'Укажите текст вопроса';
                return;
            }
            if (!this.answer?.trim()) {
                this.errorMessage = 'Укажите SQL ответ';
                return;
            }
            try {
                const { error } = await supabase
                    .from('questions')
                    .insert({
                        question: this.question,
                        correct_answer: this.answer,
                        test_id: this.test_id,
                        complexity: this.complexity
                    });
                if (error) throw error;
```

```
                emitter.emit('questionsUpdated');
                this.dialog = false;
                this.question = '';
                this.answer = '';
                this.complexity = 1;
                this.errorMessage = '';
                this.successMessage = '';
            } catch (error) {
                this.errorMessage = error.message;
            }
        },
        async do_magic() {
            this.loading = true;
            this.errorMessage = '';
            try {
                const schema = await user_db_api.get_schema(this.test_id);
                const generated_q = await
ai_api.create_question(schema.data.schema, this.complexity);
                this.question = generated_q.data.question;
                this.answer = generated_q.data.correct_sql;
            } catch (error) {
                this.errorMessage = 'Ошибка при генерации вопроса';
            }
            this.loading = false;
        },
    },
}
```

Страница `SolveTestPage.vue` — решение теста студентом (разметка трёхколоночного макета):

```
<div class="flex h-screen">
    <div class="bg-white rounded-xl m-5 ml-10 w-1/5 p-5" style="height:
calc(100%);">
        <h1 class="text-xl font-bold">Вопросы:</h1>
        <div class="flex-col">
            <div class="mt-3" v-for="q, i in questions" :key="q.id">
                <div @click="change_selected_question(q)" class="flex text-
gray-500 hover:text-blue-600">
                    <h1 class="mr-3 px-2 rounded-full font-bold"
                        :class="q == selected_question ? 'bg-blue-600 text-
white' : 'bg-gray-200'">{{ i + 1 }}</h1>
                    <h1>{{ q.question }}</h1>
                    <div class="ml-5">
                        <v-icon v-if="results.find(el => el[0] === q &&
el[1] == true)"
                            icon="mdi-check-circle" color="green"></v-icon>
                        <v-icon v-if="results.find(el => el[0] === q &&
el[1] == false)"
                            icon="mdi-close-circle" color="red"></v-icon>
                    </div>
                </div>
            </div>
            <complexity-line v-
if="selected_question" :complexity="selected_question.complexity"></
complexity-line>
        </div>
    </div>
```

```
    <div class="flex flex-col w-3/5 flex-grow">
        <div class="bg-white rounded-xl m-5 flex-grow p-5 pb-20">
            <textarea ref="sqlEditor"></textarea>
            <div class="flex justify-end border-t-2 pt-5">
                <v-btn @click="send_answer" prepend-icon="mdi-check"
color="blue-accent-4">Отправить ответ</v-btn>
                <v-btn class="ml-5" prepend-icon="mdi-play"
variant="outlined" @click="send_query">Выполнить</v-btn>
            </div>
            <ResponseTable class="mt-5" :result="response" />
        </div>
    </div>
    <div class="bg-white rounded-xl m-5 mr-10 w-1/5 p-5">
        <v-zoomer ref="er_diagram" class="h-96 rounded-xl border-2 mb-5">
            <img :src="'' + er_diagram" alt="ER
Diagram">
        </v-zoomer>
        <v-btn v-if="results.length === questions.length"
@click="send_results"
            append-icon="mdi-arrow-right" color="blue-accent-4">Завершить</
v-btn>
    </div>
</div>
```

Страница `SolveTestPage.vue` — основные методы выполнения и проверки ответа:

```
async send_query() {
    if (!this.editor) return;
    this.loading = true;
    const user_query = this.editor.getValue().replace(/\n/g, ' ');
    try {
        this.error = '';
        const response = await user_db_api.query_to_user_db(this.test_id,
user_query);
        this.response = response.data;
    } catch (error) {
        this.error = error.response?.data?.detail || error.message ||
'Ошибка выполнения запроса';
        this.response = [];
    }
    this.loading = false;
},

async change_selected_question(q) {
    if (!q) return;
    this.selected_question = q;
    this.error = '';
    this.response = [];
    try {
        this.answer = await user_db_api.query_to_user_db(this.test_id,
this.selected_question.correct_answer);
    } catch (error) {
    }
},

async send_answer() {
    if (!this.editor || !this.selected_question) return;
```

```
    try {
        const user_query = this.editor.getValue().replace(/\n/g, ' ');
        const user_answer = await user_db_api.query_to_user_db(this.test_id,
user_query)
        const isCorrect = JSON.stringify(this.answer.data?.result) ===
JSON.stringify(user_answer.data?.result);
        this.results.push([this.selected_question, isCorrect])
        const currentIndex = this.questions.indexOf(this.selected_question);
        if (currentIndex >= this.questions.length - 1) {
            this.selected_question = this.questions[0]
        } else {
            this.change_selected_question(this.questions[currentIndex + 1])
        }
        this.editor.setValue('')
        this.error = '';
        this.response = [];
    } catch (error) {
        this.error = error.response?.data?.detail || error.message ||
'Ошибка отправки ответа';
        this.response = [];
    }
},

async send_results() {
    if (!this.currentUser) {
        this.error = 'Пользователь не авторизован';
        return;
    }
    const data = {
        "user_id": this.currentUser.id,
        "test_id": this.test_id,
        "correct_answers": this.results.filter(pair => pair[1] ===
true).length,
        "number_of_questions": this.questions.length
    }
    const { error } = await supabase.from('results').insert(data);
    if (error) {
        this.error = 'Ошибка сохранения результата: ' + error.message;
        return;
    }
    this.$router.push("/")
},

async loadERDiagram() {
    try {
        const erResult = await user_db_api.create_er_diagram(this.test_id);
        this.er_diagram = erResult.data.er_diagram;
    } catch (error) {
    }
}
```

Страница `TestPage.vue` — построение статистики прохождений теста (логика):

```
async fetch_chart_data(res) {
    this.chart_data = {
        labels: Array.from(Array(res[0].number_of_questions + 1).keys(), x
=> (x).toString()),
```

```
        datasets: [
            {
                backgroundColor: ['#3662E3', '#436CE5', '#2353E1',
'#1A44C2', '#7D99ED', '#A8BBF3', '#13328D'],
                data: Array.from(Array(res[0].number_of_questions +
1).keys(), x => (x))
                    .map(grade => res.map(result => result.correct_answers)
                        .filter(result => result === grade).length)
            }
        ]
    };
},

async mounted() {
    emitter.on('questionsUpdated', () => {
        this.fetch_questions();
    });
    this.fetch_questions();
    const { data: resultsData, error } = await supabase
        .from('results')
        .select(`*, user:users(name, "group")`)
        .eq('test_id', this.test_id)
        .order('created_at', { ascending: false });
    if (!error && resultsData && resultsData.length > 0) {
        this.results = resultsData.map(item => ({
            ...item,
            expand: {
                user_id: {
                    name: item.user?.name || 'Unknown',
                    group: item.user?.['group'] || 'Unknown'
                }
            }
        }));
        this.fetch_chart_data(this.results);
        this.draw = true;
    }
}
```

Компонент `SqlEditor.vue` — редактор SQL на основе CodeMirror:

```
export default {
    mounted() {
        const sqlEditor = this.$refs.sqlEditor;
        this.editor = CodeMirror.fromTextArea(sqlEditor, {
            mode: 'text/x-sql',
            lineNumbers: true,
            autofocus: true,
        });
    },
    beforeUnmount() {
        if (this.editor) {
            this.editor.toTextArea();
        }
    },
};
```