

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**НЕЙРОСЕТЕВОЙ АССИСТЕНТ С
ИНТЕРАКТИВНОЙ ПАНЕЛЬЮ**

Студент Д.А. Дулатов 11.06.2026
(подпись, дата)

Руководитель Т.Ю. Леонова 11.06.2026
(подпись, дата)

Нормоконтролёр А.С. Бычков 11.06.2026
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от
22.05.2026 г. № 14)

Заведующий кафедрой М.А. Митрохин
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 19.06.2026 № 38)

Секретарь ГЭК Е.И. Гурин
(подпись)

Пенза, 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» 04 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Дулатов Денис Александрович гр.22ВВП2 факультета вычислительной техники.

2. Тема работы: «Нейросетевой ассистент с интерактивной панелью»

Тема утверждена приказом ПГУ № 347/о от "7" апреля 2026 г.

3. Руководитель бакалаврской работы к.т.н., доцент Леонова Т.Ю.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Нейросетевой ассистент с интерактивной панелью», предназначенное для ведения структурированных диалогов с искусственным интеллектом и визуальной организации полученной информации. Приложение должно быть спроектировано с применением клиент-серверной архитектуры. Серверная часть должна обеспечивать обработку запросов, взаимодействие с нейросетевой моделью и хранение данных. Клиентская часть должна представлять собой веб-приложение с графическим интерфейсом, доступным через браузер. Система должна включать в себя функционал создания и удаления рабочих областей (диалогов), отправки текстовых запросов к нейросетевой модели, потокового отображения генерируемых ответов в режиме реального времени, а также визуального управления информационными карточками (перемещение в двухмерном пространстве, группировка). Клиентская программа должна позволять пользователю управлять сессиями диалогов, просматривать историю запросов и ответов, а также структурировать полученную информацию в виде перемещаемых элементов на интерактивной панели.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить аналоги существующих систем для визуальной организации информации и взаимодействия с нейросетевыми ассистентами, изучить актуальные способы интеграции открытых языковых моделей в веб-приложения, рассмотреть методы потоковой передачи данных через WebSocket-соединения. Также необходимо изучить способы проектирования и разработки клиент-серверного приложения с применением REST API методологии, функционал асинхронного веб-фреймворка FastAPI, возможности библиотеки React для построения интерактивных пользовательских интерфейсов.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	48	07.04.2026 – 13.04.2026	hech
2	Анализ требований	36	14.04.2026 – 18.04.2026	hech
3	Выбор технологий, применяемых при проектировании и разработке	40	19.04.2026 – 24.04.2026	hech
4	Создание графического интерфейса	12	25.04.2026 – 26.04.2026	hech
5	Разработка приложения	168	27.04.2026 – 18.05.2026	hech
6	Оформление пояснительной записки	20	19.05.2026 – 21.05.2026	hech

Дата выдачи задания "04" апреля 2026 г.

Руководитель бакалаврской работы hech Леонова Т.Ю.
(подпись, дата)

Задание к исполнению принял студент SD Дулатов Д.А.
(подпись, дата)

Работу к защите допустить
Декан факультета 16.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 87 листов, 18 рисунков, 5 таблиц, 4 источника и 2 приложения.

**НЕЙРОСЕТЕВОЙ АССИСТЕНТ, ИНТЕРАКТИВНАЯ ПАНЕЛЬ,
ЯЗЫКОВАЯ МОДЕЛЬ, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА, ВЕБ-
ПРИЛОЖЕНИЕ, REACT, FASTAPI, ПОТОКОВАЯ ПЕРЕДАЧА ДАННЫХ.**

Цель работы – разработать веб-приложение, объединяющее нейросетевого ассистента на базе открытой языковой модели и интерактивную панель для визуальной организации информации в процессе диалога.

Объектами разработки являются:

- серверное приложение для обработки запросов пользователя, взаимодействия с нейросетевой моделью, хранения данных о диалогах и информационных карточках в базе данных, а также организации потоковой передачи генерируемых ответов;

- клиентское веб-приложение, предоставляющее графический интерфейс для создания и управления диалогами, отправки запросов к нейросетевому ассистенту и визуального управления информационными карточками на интерактивной панели.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: разработана серверная часть на базе асинхронного фреймворка FastAPI, обеспечивающая взаимодействие с языковой моделью и базой данных SQLite; реализовано REST API и WebSocket-соединение для обмена данными с клиентской частью; создано клиентское веб-приложение на базе библиотеки React.

					ПГУ 1.090301.36.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Дулатов Д.А.		11.06.26	Нейросетевой ассистент с интерактивной панелью Пояснительная записка		
Провер		Леонова Т.Ю.		11.06.26			
Ил. Контр		Бычков А.С.		11.06.26			
Утверд		Митрахин М.А.		11.06.26			
					Лит.	Лист	Листов
						2	46
					ФВТ, каф. ВТ		
					гр. 22ВВП2		

Содержание

Перечень сокращений и обозначений	7
Введение	8
1 Нейросетевой ассистент с интерактивной панелью	9
1.1 Анализ предметной области	9
1.1.1 Обзор аналогов	10
1.1.2 Анализ требований к разработке	11
1.1.3 Выбор архитектуры приложения.....	13
1.1.4 Выбор языка программирования	14
1.1.5 Выбор фреймворка для разработки серверного приложения.....	15
1.1.6 Выбор библиотеки для пользовательского интерфейса.....	17
1.1.7 Выбор системы управления базами данных	19
1.1.8 Выбор среды разработки	20
1.2 Проектирование и разработка.....	22
1.2.1 Реализация взаимодействия с языковой моделью	22
1.2.2 Разработка базы данных	25
1.2.3 Реализация системы категоризации ответов	26
1.2.4 Разработка клиентской части.....	28
1.2.5 Разработка собственного REST API.....	30
1.2.6 Разработка WebSocket-взаимодействия	31
1.2.7 Обработка и сохранение контекста диалога	33
1.3 Анализ результатов.....	34
1.3.1 Тестирование серверной части	35
1.3.2 Тестирование клиентской части	38
2 Безопасность жизнедеятельности.....	41
2.1 Характеристика вредных факторов при работе с компьютером	41
2.2 Классификация помещения по категориям опасности.....	42
2.2.1 Пожарная безопасность	42

2.2.2 Электробезопасность	42
2.3 Санитарно-гигиенические требования к организации рабочего места	43
2.4 Режим труда и отдыха при работе с ЭВМ	44
Заключение	45
Список использованных источников.....	46
Приложение А.....	47
Приложение В.....	50
Приложение Г	83

Перечень сокращений и обозначений

В настоящей работе применяются следующие сокращения и обозначения

БД – база данных

ООП – объектно-ориентированное программирование

ПО – программное обеспечение

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс приложения

CI/CD (Continuous Integration / Continuous Delivery) – непрерывная интеграция и доставка

CRUD (Create, Read, Update, Delete) – создание, чтение, обновление, удаление

CSS (Cascading Style Sheets) – каскадные таблицы стилей

HTML (HyperText Markup Language) – язык гипертекстовой разметки

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста

IDE (Integrated Development Environment) – интегрированная среда разработки

LLM (Large Language Model) – большая языковая модель

ORM (Object-Relational Mapping) – объектно-реляционное отображение

REST (Representational State Transfer) – передача репрезентативного состояния

SPA (Single Page Application) – одностраничное веб-приложение

SQL (Structured Query Language) – язык структурированных запросов

SSH (Secure Shell) – протокол защищённого удалённого доступа

WebSocket – протокол двунаправленной связи поверх HTTP

Введение

Развитие технологий искусственного интеллекта привело к появлению больших языковых моделей, способных решать широкий спектр интеллектуальных задач. При этом возрастают объёмы данных, с которыми приходится работать специалистам в самых разных областях. Традиционные инструменты для ведения заметок не справляются с наглядным представлением разветвлённых цепочек рассуждений и не имеют встроенных средств для взаимодействия с нейросетевыми ассистентами.

Актуальность темы обусловлена растущей потребностью в программных средствах, объединяющих возможности современных языковых моделей и удобные визуальные интерфейсы для организации информации. Существующие решения либо ориентированы на облачные API коммерческих провайдеров, либо представляют собой сложные профессиональные пакеты, недоступные широкому кругу пользователей из-за высокой стоимости и избыточной функциональности. Открытые же языковые модели, пригодные для локального развёртывания, как правило, не имеют развитого пользовательского интерфейса и требуют от пользователя определённых технических навыков для их настройки и эксплуатации.

Цель работы — разработка клиент-серверного веб-приложения, объединяющего нейросетевого ассистента на базе открытой языковой модели и интерактивную панель для визуальной организации информации. Для этого необходимо разработать серверную часть с поддержкой языковой модели, клиентскую часть с интерактивной панелью, настроить потоковую передачу ответов и обеспечить доступ через браузер.

На рынке представлено мало решений, сочетающих удобный интерфейс для наглядной организации информации и возможность полноценного диалога с нейросетевым ассистентом в рамках единого приложения. Разрабатываемое приложение призвано заполнить эту нишу, предоставив пользователям доступный инструмент для работы с информацией.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		6

1 Нейросетевой ассистент с интерактивной панелью

1.1 Анализ предметной области

В эпоху информационной перегрузки специалисты в самых разных областях, таких как аналитика данных, менеджмент, научные исследования и образование, сталкиваются с необходимостью быстрой обработки и осмысления больших объёмов слабоструктурированной информации. Традиционные линейные форматы представления данных (текстовые документы, электронные таблицы, линейные заметки) не позволяют эффективно проследивать логические связи между разрозненными тезисами, концепциями или идеями.

При работе над сложным проектом, изучении новой темы или подготовке научной публикации пользователю требуется не просто фиксировать отдельные факты и сведения, а осмысленно систематизировать накопленный материал, представляя его в наглядной и удобной для восприятия форме. Возникает потребность в инструменте, который визуально отображает ход взаимодействия с искусственным интеллектом в виде набора динамических информационных блоков, размещённых на интерактивной панели и доступных для свободного перемещения.

Кроме визуального структурирования, остро стоит вопрос автоматизации когнитивных задач. Существующие облачные решения с нейросетевым функционалом предоставляются по платной подписке и не имеют развитых средств визуальной организации информации, в то время как открытые языковые модели, пригодные для локального развёртывания, лишены удобного веб-интерфейса и требуют от пользователя технических навыков для их эксплуатации. Таким образом, существует потребность в создании программного продукта, объединяющего интерактивную панель для визуальной работы с информацией и нейросетевого ассистента в рамках единой клиент-серверной архитектуры, доступной через браузер.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		7

1.1.1 Обзор аналогов

Рынок программного обеспечения для организации информации и взаимодействия с искусственным интеллектом представлен широким спектром продуктов. Условно существующие решения можно разделить на две категории: облачные сервисы для совместной работы и профессиональные пакеты для локального развёртывания [1].

К первой категории относятся популярные онлайн-доски, такие как Miro и FigJam, а также системы ведения персональных баз знаний, включая Obsidian и Notion. Данные платформы предоставляют пользователю удобное визуальное пространство для размещения карточек, текстовых блоков, изображений и других элементов, позволяя гибко организовывать информацию в двухмерной плоскости. Однако их функционал в части взаимодействия с искусственным интеллектом носит второстепенный, вспомогательный характер и реализован исключительно как дополнение к основному продукту. Нейросетевые возможности, как правило, сводятся к генерации текста по короткому запросу или автоматическому составлению краткой сводки, при этом пользователь лишён возможности вести полноценный, контекстно-зависимый диалог с ассистентом. Средства визуализации истории общения с нейросетью и управления разветвлёнными цепочками рассуждений в таких сервисах либо отсутствуют, либо развиты крайне слабо. Таким образом, пользователь не может рассматривать нейросетевого ассистента как центральный элемент своего рабочего процесса.

Ко второй категории относятся специализированные профессиональные пакеты для аналитической работы, поддерживающие развёртывание на локальном оборудовании. К ним относятся такие решения, как i2 Analyst's Notebook, применяемый для визуализации связей между объектами, а также различные корпоративные BI-системы с функциями построения интерактивных отчётов и дашбордов. Данные программные продукты ориентированы преимущественно на крупные организации и государственные

структуры, что определяет их ключевые недостатки применительно к рядовому пользователю: функциональная избыточность требует длительного изучения документации и освоения сложного интерфейса, а высокая стоимость лицензий с обязательной ежегодной платой за поддержку делает их экономически недоступными для индивидуальных специалистов, малого бизнеса или образовательных учреждений.

Сравнение возможностей алгоритмов разных компаний представлена в таблице 1.1.

Таблица 1.1 – Сравнение систем для работы с интерактивными панелями и искусственным интеллектом

Критерии	Облачные доски (Miro и др.)	Профессиональные аналитические пакеты
Визуальное управление элементами	Да, на высоком уровне	Ограничено или устаревший интерфейс
Интеграция с локальной LLM	только облачные API	Нет или требует сложной настройки
Гибкость интерфейса	Высокая, интуитивно понятен	Низкая, требуется длительное освоение
Простота развертывания	Не требуется	Сложная установка и лицензирование

1.1.2 Анализ требований к разработке

По результатам изучения предметной области и принимая во внимание обнаруженные недостатки действующих облачных платформ и специализированных аналитических комплексов, был сформирован перечень следующих функциональных требований:

- создание нового диалога для начала общения с нейросетевым ассистентом;
- удаление существующего диалога из списка доступных;
- смена текущего диалога для продолжения ранее начатой беседы;
- отправка запроса к нейросетевой модели и получение ответа в режиме реального времени;
- прерывание генерации ответа до его завершения.

На основе выделенных функциональных требований построена диаграмма вариантов использования, представленная на рисунке 1.



Рисунок 1 – Диаграмма вариантов использования

К разрабатываемому приложению также предъявляются следующие технические требования:

- клиент-серверная архитектура с разделением на фронтенд и бэкенд.
- доступ к приложению через веб-браузер без установки дополнительного ПО.
- взаимодействие клиента и сервера посредством REST API и WebSocket.
- интеграция с открытой языковой моделью (LLM) без обращения к сторонним облачным API.
- хранение данных о диалогах и элементах в реляционной базе данных.

1.1.3 Выбор архитектуры приложения

Вычислительная нагрузка, возникающая при работе языковых моделей, предъявляет повышенные требования к аппаратному обеспечению. Совмещение пользовательского интерфейса и модели машинного обучения в рамках одного настольного приложения не является оптимальным решением для организации удаленного многопользовательского доступа. В связи с этим для реализации проекта была выбрана клиент-серверная архитектура, позволяющая разделить зоны ответственности между компонентами системы.

Серверная часть представляет собой бэкенд-приложение, функционирующее по принципам REST API и WebSocket-сервера. Сервер разворачивается на выделенной машине, обладающей достаточной вычислительной мощностью для обработки запросов, и предоставляет доступ к своему функционалу по стандартизированным протоколам HTTP и WebSocket.

Клиентская часть реализована как одностраничное веб-приложение, загружаемое в браузере пользователя. Такой подход снимает необходимость установки специализированного программного обеспечения на каждое клиентское устройство и обеспечивает кроссплатформенность: доступ к системе возможен с персональных компьютеров, планшетов и мобильных устройств независимо от установленной на них операционной системы. Клиент отвечает за рендеринг интерактивной панели, обработку действий пользователя (перемещение элементов, ввод текста) и обмен данными с сервером.

Взаимодействие между клиентом и сервером организовано следующим образом: для операций создания, чтения, обновления и удаления информационных блоков используются REST-запросы, а для отображения генерации текста нейросетью в реальном времени устанавливается постоянное WebSocket-соединение, по которому токены передаются по мере их появления на выходе модели. Архитектура серверной части представлена на рисунке 2.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

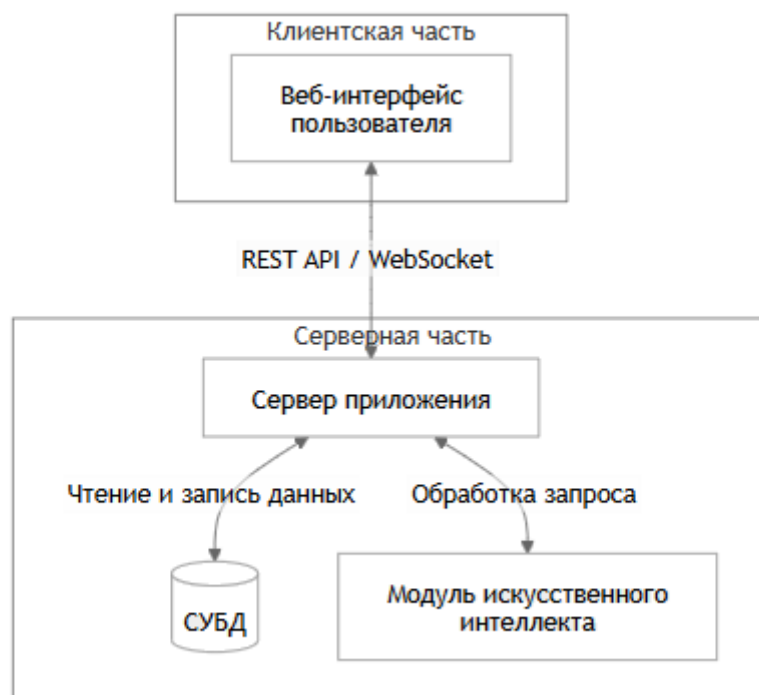


Рисунок 2 – Архитектура веб-приложения

1.1.4 Выбор языка программирования

Выбор конкретных технологий и языков программирования осуществлялся на основе критериев производительности, удобства поддержки кода и наличия необходимых библиотек для решения поставленных задач.

Для реализации серверной части был выбран язык Python 3.13 в связке с асинхронным веб-фреймворком FastAPI. Python является стандартом в области машинного обучения. Богатая экосистема библиотек, включая средства для работы с моделями семейства Llama, позволяет ускорить процесс разработки и сконцентрировать усилия на реализации ключевых процессов программы. FastAPI, со своей стороны, включает готовые механизмы для неблокирующих вычислений, постоянных WebSocket-каналов связи и самостоятельного формирования интерактивных справочных страниц в стандарте OpenAPI (Swagger).

Для реализации клиентской интерактивной панели выбран язык TypeScript с библиотекой React. Строгая типизация TypeScript снижает вероятность ошибок на этапе разработки и облегчает рефакторинг кода.

Сборка проекта осуществляется с помощью инструмента Vite, обеспечивающего высокую скорость обновления модулей в процессе отладки и оптимизированную финальную сборку. Для реализации свободного перемещения элементов по рабочей области используется специализированная библиотека, предоставляющая готовые компоненты для рендеринга узлов, обработки событий перетаскивания, масштабирования и панорамирования. Визуальное оформление интерфейса выполнено с использованием CSS-фреймворка Tailwind CSS, позволяющего быстро создавать адаптивные и стилистически единообразные компоненты.

Сведения о беседах, информационных блоках и их координатах сохраняются средствами встраиваемой реляционной СУБД SQLite. Подобный подход даёт возможность содержать все данные в границах одного локального файла, что облегчает создание резервных копий и миграцию информации при возникновении такой необходимости. Обращение к базе выполняется через объектно-реляционную библиотеку SQLAlchemy, которая предлагает разработчику понятный интерфейс для составления запросов, построенный на принципах объектно-ориентированного программирования.

1.1.5 Выбор фреймворка для разработки серверного приложения

Для построения серверной составляющей веб-приложений на языке Python применяется целый ряд известных инструментальных каркасов, среди которых наибольшее распространение получили Django, Flask и FastAPI. Все они характеризуются собственными архитектурными чертами, сильными сторонами и ограничениями.

Django является полнофункциональным веб-фреймворком, предоставляющим разработчику широкий набор встроенных инструментов: собственную ORM, систему шаблонизации, панель администратора, средства аутентификации и множество других компонентов. Однако такая функциональная насыщенность приводит к высокой связанности компонентов и избыточности при разработке легковесных API-сервисов. Кроме того, Django изначально проектировался для синхронной обработки запросов, что

ограничивает его производительность при работе с длительными операциями, такими как инференс нейросетевых моделей.

Flask представляет собой микрофреймворк, предоставляющий только базовый функционал для обработки HTTP-запросов. Он отличается простотой освоения и гибкостью, позволяя разработчику самостоятельно выбирать необходимые расширения для реализации требуемой функциональности. Однако отсутствие встроенной поддержки асинхронных операций и необходимость ручной настройки WebSocket-соединений делают Flask менее удобным для задач, связанных с потоковой передачей данных в реальном времени [3].

FastAPI является современным асинхронным веб-фреймворком, специально спроектированным для создания высокопроизводительных API. Ключевыми преимуществами FastAPI являются встроенная поддержка асинхронных операций на основе библиотеки Starlette, автоматическая генерация интерактивной документации OpenAPI (Swagger), а также нативная работа с WebSocket-соединениями. Эти особенности делают FastAPI оптимальным выбором для приложений, требующих одновременной обработки множества запросов и организации потокового обмена данными, что критически важно при взаимодействии с нейросетевыми моделями.

В таблице 1.2 представлено сравнение рассмотренных фреймворков.

Учитывая требования проекта к асинхронной обработке запросов и необходимости поддержки потоковой передачи данных через WebSocket, для реализации серверной части был выбран фреймворк FastAPI.

Таблица 1.2 – Сравнение фреймворков

Название	<i>ORM</i>	Асинхронная обработка	Поддержка WebSocket	Автодокументация API
<i>Django</i>	Да	Ограничена	Через каналы	Через расширения
<i>Flask</i>	Нет	Нет	Через расширения	Нет
<i>FastAPI</i>	Нет	Да, нативная	Да, встроенная	Да, Swagger

1.1.6 Выбор библиотеки для пользовательского интерфейса

Для разработки клиентской части веб-приложения, представляющей собой интерактивную панель с возможностью свободного перемещения элементов, требуется библиотека, обеспечивающая высокую производительность рендеринга и удобство работы с динамическим интерфейсом. На сегодняшний день существует три основных подхода к построению веб-интерфейсов: использование чистого JavaScript, применение императивных библиотек (таких как jQuery) и использование компонентно-ориентированных фреймворков. Приблизительная схема запроса к БД через *ORM* представлена на рисунке 2.

Чистый JavaScript и императивные библиотеки требуют ручного управления DOM-деревом и состоянием интерфейса, что при возрастании сложности приложения приводит к запутанности кода и снижению его поддерживаемости. Данный подход не рекомендуется для проектов с развитым пользовательским интерфейсом.

Среди компонентно-ориентированных решений наиболее популярны три библиотеки: React, Vue.js и Angular. Angular представляет собой полноценный фреймворк со встроенными средствами маршрутизации, управления состоянием и работы с формами. Однако его высокая сложность и крутая кривая обучения делают его избыточным для задач данного проекта. Vue.js отличается простотой освоения и гибкостью, но имеет меньшее сообщество разработчиков и меньшее количество готовых библиотек для

визуализации сложных интерактивных элементов по сравнению с React, поэтому React остаётся наиболее прагматичным выбором благодаря огромной экосистеме и проверенным решениям для визуализации.

Ключевыми преимуществами React являются компонентный подход, обеспечивающий высокую производительность интерфейса, а также обширная экосистема решений. Для реализации функционала интерактивной панели в экосистеме React существует библиотека React Flow, предоставляющая готовые компоненты для рендеринга перемещаемых узлов, обработки событий перетаскивания, масштабирования и панорамирования рабочей области.

В таблице 1.3 представлено сравнение рассмотренных библиотек.

Таблица 1.3 – Сравнение библиотек для построения пользовательского интерфейса

Критерий	React	Vue.js	Angular
Компонентный подход	Да	Да	Да
Виртуальный DOM	Да	Да	Нет (инкрементальный)
Экосистема библиотек	Очень обширная	Средняя	Встроенные средства
Поддержка интерактивных панелей	Да (React Flow)	Ограничено	Нет

Учитывая наличие специализированной библиотеки React Flow для построения интерактивных перемещаемых панелей, а также широкие возможности по стилизации компонентов, для реализации клиентской части приложения была выбрана библиотека React в связке с языком TypeScript.

1.1.7 Выбор системы управления базами данных

Для хранения информации о диалогах, карточках и их позициях в рамках клиент-серверного приложения требуется система управления базами данных (СУБД), обеспечивающая надёжное структурированное хранение данных и удобный доступ к ним из серверной части.

Современные СУБД делятся на два основных класса: реляционные (SQL) и нереляционные (NoSQL). Нереляционные базы данных, такие как MongoDB, хранят информацию в виде документов произвольной структуры, что обеспечивает гибкость при изменении схемы данных. Однако в рамках разрабатываемого приложения структура данных является строго определённой и редко изменяемой, поэтому преимущества NoSQL-подхода не являются критичными.

Среди реляционных СУБД наиболее распространены PostgreSQL, MySQL и SQLite. PostgreSQL является мощной промышленной СУБД с поддержкой сложных типов данных, хранимых процедур и высокой конкурентной нагрузки. MySQL также широко применяется в веб-разработке и отличается высокой производительностью на операциях чтения. Обе эти СУБД требуют развёртывания отдельного серверного процесса и его администрирования, что увеличивает сложность установки и сопровождения системы.

SQLite представляет собой легковесную встраиваемую реляционную СУБД, не требующую отдельного серверного процесса. Вся база данных хранится в одном файле на диске, что упрощает развёртывание и перенос данных. SQLite поддерживает стандартный синтаксис SQL и транзакции ACID, обеспечивая необходимый уровень надёжности для приложений с умеренной нагрузкой. Взаимодействие с SQLite из серверной части на Python осуществляется через ORM-библиотеку SQLAlchemy, предоставляющую удобный интерфейс для выполнения запросов.

В таблице 1.4 представлено сравнение рассмотренных СУБД.

					ПГУ 1.090301.36.001 ПЗ	Лист 17
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1.4 – Сравнение систем управления базами данных

Критерий	SQLite	PostgreSQL	MySQL
Требуется отдельный сервер	Нет	Да	Да
Поддержка ACID	Да	Да	Да
Сложность развёртывания	Легкая	Средняя	Средняя
Поддержка SQLAlchemy	Да	Да	Да
Подходит для умеренных нагрузок	Да	Да	Да

Учитывая требования к простоте развёртывания и отсутствию необходимости в администрировании отдельного сервера баз данных, для реализации проекта была выбрана СУБД SQLite.

1.1.8 Выбор среды разработки

Разработка современного программного обеспечения включает в себя не только написание исходного кода, но и отладку, управление зависимостями, работу с системой контроля версий и многие другие задачи. Интегрированная среда разработки (IDE) объединяет эти инструменты в едином интерфейсе, повышая производительность труда программиста.

Для разработки на Python и TypeScript существует несколько популярных сред. PyCharm, разработанный компанией JetBrains, предоставляет мощные средства для работы с Python-проектами, включая интеллектуальное автодополнение, отладчик и встроенную поддержку виртуальных окружений. Visual Studio Code отличается высокой производительностью и расширяемостью за счёт плагинов, поддерживая как Python, так и JavaScript/TypeScript на высоком уровне. Atom и Sublime Text представляют собой более легковесные редакторы, однако их функциональность существенно уступает полноценным IDE при работе над комплексными проектами [4].

Дополнительным преимуществом SQLite является её широкая распространённость и стабильность: библиотека SQLite входит в состав стандартной поставки Python, что исключает необходимость установки дополнительных драйверов и упрощает развёртывание приложения на любой платформе. Данная СУБД активно применяется в прототипировании, небольших веб-сервисах и проектах с числом одновременных подключений, не превышающим нескольких десятков, что полностью соответствует характеру нагрузки разрабатываемого приложения.

Следует также отметить, что SQLite поддерживает режим WAL (Write-Ahead Logging), позволяющий совмещать параллельное чтение и запись без блокировок, что положительно сказывается на отзывчивости серверной части при одновременной работе нескольких пользователей. В сочетании с ORM-библиотекой SQLAlchemy разработчик получает возможность строить запросы на языке Python без написания SQL-кода, а при необходимости — выполнять и прямые SQL-команды, что обеспечивает гибкость при оптимизации наиболее нагруженных участков кода.

Ключевым фактором при выборе среды для данного проекта является необходимость одновременной работы с двумя языками программирования (Python на сервере и TypeScript на клиенте) в рамках единого монорепоzitория. Visual Studio Code обеспечивает первоклассную поддержку обоих языков через официальные расширения, позволяет работать с одним проектом, содержащим разнородный код, и имеет встроенную интеграцию с системой контроля версий Git. PyCharm, несмотря на превосходную поддержку Python, менее удобен при работе с современными JavaScript-фреймворками в том же проекте.

Учитывая необходимость унифицированной работы с клиентской и серверной частями в рамках единой кодовой базы, в качестве среды разработки была выбрана Visual Studio Code.

1.2 Проектирование и разработка

Приложение состоит из серверной и клиентской частей. Сервер обрабатывает запросы к нейросетевой модели, управляет диалогами и хранит информацию о пользователях, темах и карточках. Клиентами могут выступать веб-браузер, Postman или клиентская программа.

Набор возможностей пользователей включает: регистрацию и авторизацию в системе, создание и удаление диалогов, отправку запросов к нейросетевому ассистенту с получением потоковых ответов, автоматическую категоризацию ответов по тематическим группам, визуальное управление карточками на интерактивной графовой панели, создание связей между карточками, а также перемещение карточек между тематическими группами.

Перед началом работы сервера производится инициализация всех компонентов и модулей системы: загружается языковая модель в оперативную память, инициализируется компонент для работы с базой данных, создаются таблицы при их отсутствии. В это же время производится подключение к СУБД SQLite.

Важным нюансом является то, что данные о диалогах, темах и карточках сохраняются в базе данных автоматически при каждом взаимодействии пользователя с ассистентом, а позиции элементов на интерактивной панели сохраняются в локальном хранилище браузера пользователя.

1.2.1 Реализация взаимодействия с языковой моделью

Для работы с языковой моделью была использована открытая и свободно распространяемая библиотека «llama-cpp-python». Она предоставляет возможность локального запуска больших языковых моделей в формате GGUF без обращения к сторонним облачным API. Для интеграции модели в асинхронный веб-сервер на FastAPI был реализован модуль chat.py, через который производится всё взаимодействие с нейросетью посредством WebSocket-соединения.

Метод `websocket_chat` принимает пользовательский запрос, формирует системный промпт с перечнем существующих тем диалога и передаёт его модели. Ответ модели в потоковом режиме передаётся клиенту по мере генерации токенов. После завершения генерации из ответа извлекается название темы, создаётся новая карточка в базе данных с привязкой к соответствующей теме. При необходимости создаётся новая тематическая группа.

Методы `get_board_topics` и `get_system_prompt` необходимы для формирования контекста диалога и передачи модели актуального перечня существующих тем, что обеспечивает корректную категоризацию новых ответов.

После создания диалога и отправки первого запроса к нейросетевому ассистенту автоматически формируется структура диалога: сохраняется текст запроса пользователя, полный ответ модели и название тематической группы, к которой был отнесён ответ. Каждая карточка связывается с соответствующей темой, а при появлении новых ответов в рамках той же темы они добавляются в существующую группу.

При повторных запросах пользователя в рамках одного диалога осуществляется поиск подходящей тематической группы среди существующих. Новые карточки добавляются в соответствующую группу, а существующие данные не обновляются. Связи между карточками, установленные пользователем вручную через интерфейс, сохраняются в базе данных. Таким образом обеспечивается накопление и структурирование информации в рамках каждого диалога с сохранением всех взаимосвязей.

После добавления пользователя в систему сразу сохраняются его учётные данные: логин и хэшированный пароль. Идентификатор пользователя связывается со всеми создаваемыми им диалогами. При отправке запроса к нейросетевому ассистенту производится сохранение текста запроса, полного ответа модели и названия тематической группы, к которой был отнесён ответ.

Таким образом собирается полная история взаимодействия в рамках каждого диалога с сохранением всех логических взаимосвязей. Стартовая страница документации API представлена на рисунке 3

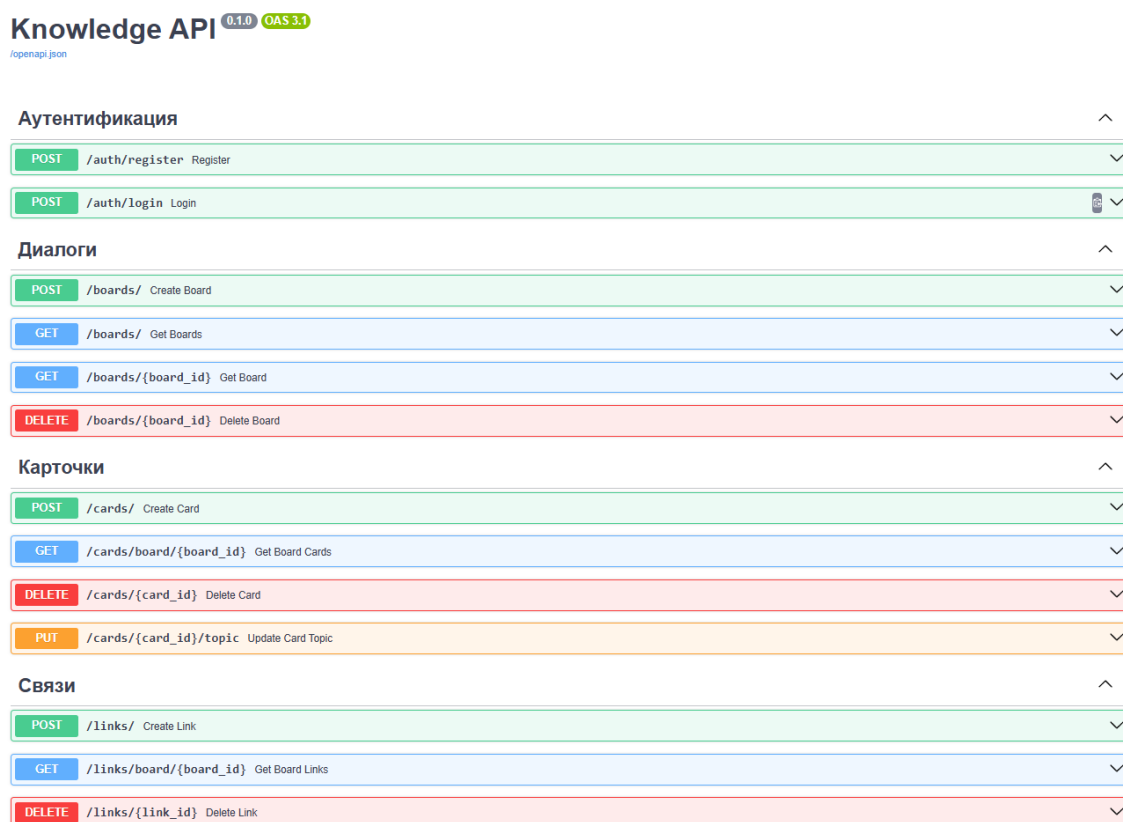


Рисунок 3 – Стартовая страница приложения FastAPI (Swagger)

Добавленные модули-роутеры:

- auth.py - для регистрации и авторизации пользователей в системе;
- boards.py - для управления диалогами (создание, удаление, получение списка);
- cards.py - для работы с карточками диалога (создание, удаление, обновление темы);
- links.py - для создания и удаления связей между карточками;
- chat.py - для взаимодействия с языковой моделью через WebSocket-соединение.

Конфигурация сервера задаётся в файле database.py. Содержимое файла определяет параметры подключения к базе данных SQLite через SQLAlchemy.

1.2.2 Разработка базы данных

Для хранения данных приложения выбрана реляционная система управления базами. Для сохранения информации приложения задействована реляционная СУБД SQLite. Данное решение продиктовано лёгкостью начальной установки — база данных хранится в одном файле на диске и не требует установки и настройки отдельного серверного процесса. SQLite поддерживает стандартный синтаксис SQL и транзакции ACID, что обеспечивает необходимый уровень надёжности для приложений с умеренной нагрузкой.

Взаимодействие с базой данных осуществляется через ORM-библиотеку SQLAlchemy, которая позволяет не писать прямые SQL-запросы на создание и изменение данных. Структура таблиц определяется классами моделей на языке Python, а миграция схемы выполняется автоматически при запуске приложения вызовом метода `Base.metadata.create_all()`. Такой подход ускоряет разработку и снижает вероятность ошибок, связанных с ручным написанием SQL-кода.

Спроектированная база данных включает пять таблиц: `users` — учётные записи пользователей, `boards` — диалоги (сессии общения с ассистентом), `topics` — тематические группы, `cards` — карточки с запросами и ответами, `links` — связи между карточками. Каждый пользователь может создавать несколько диалогов, каждый диалог содержит множество тематических групп и карточек. Карточка может принадлежать одной теме и иметь несколько входящих и исходящих связей с другими карточками. Схема базы данных представлена на рисунке 4.

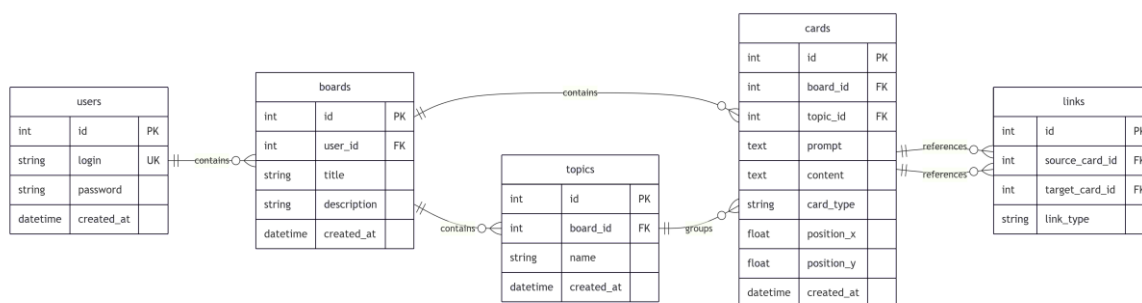


Рисунок 4 – Схема базы данных

1.2.3 Реализация системы категоризации ответов

Категоризация ответов нейросетевого ассистента по тематическим группам является достаточно сложной задачей в области обработки естественного языка, так как требует определения семантической близости произвольных запросов пользователя к существующим темам диалога.

Наиболее эффективным подходом к решению данной задачи является использование самой языковой модели для определения тематики. Модель Saiga Gemma 3, обученная на русскоязычных данных, способна не только генерировать ответы, но и определять их тематическую принадлежность на основе инструкций в системном промпте. При формировании запроса в системный промпт включается актуальный перечень существующих тем диалога. Модель получает инструкцию отнести ответ к одной из существующих тем либо предложить новую тему при отсутствии подходящей.

Выбранный подход позволяет избежать необходимости реализации отдельного классификатора или дополнительных алгоритмов машинного обучения для определения тематики. Поскольку модель уже обладает пониманием смысла как запроса пользователя, так и собственного ответа, она способна оценить, к какой из существующих тем относится текущий фрагмент беседы, с высокой точностью. Это упрощает архитектуру серверной части и снижает вычислительную нагрузку, так как категоризация выполняется в рамках того же запроса, что и генерация ответа.

После получения ответа от модели производится извлечение названия темы с помощью регулярных выражений. В случае если указанная тема отсутствует в базе данных для текущего диалога, создаётся новая тематическая группа. Карточка с ответом связывается с соответствующей темой, благодаря чему все ответы по одной тематике автоматически группируются в рамках интерактивной панели. Алгоритм категоризации ответов представлен на рисунке 5.

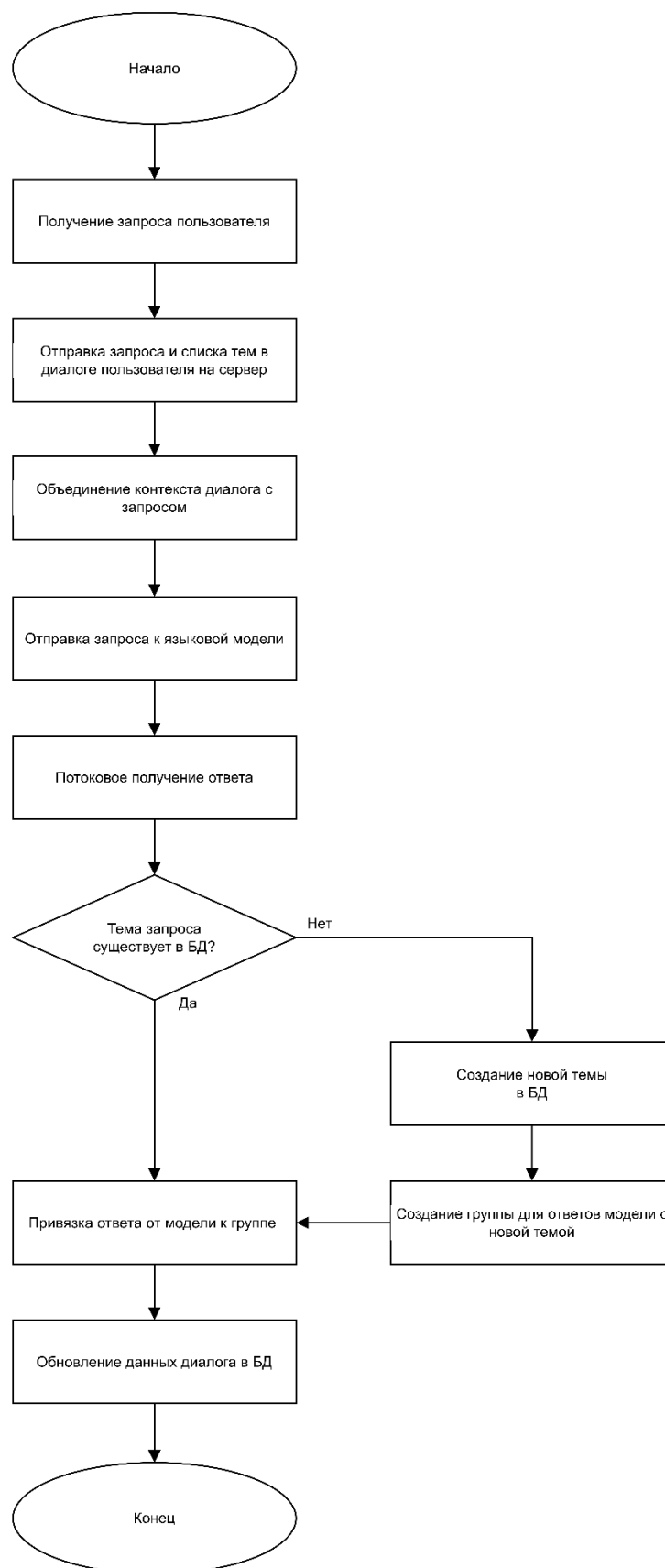


Рисунок 5 – Алгоритм категоризации ответов нейросетевого ассистента

1.2.4 Разработка клиентской части

Разработка пользовательского интерфейса является одной из важнейших задач создания веб-приложения. Интерфейс должен быть одновременно функциональным и интуитивно понятным, обеспечивая пользователю удобное взаимодействие с нейросетевым ассистентом и визуальную организацию информации.

Для построения клиентской части были использованы две основные библиотеки: React — для создания компонентной структуры интерфейса, и React Flow — для реализации интерактивной панели с возможностью свободного перемещения элементов. React Flow предоставляет готовые компоненты для рендеринга узлов, обработки событий перетаскивания, масштабирования и панорамирования рабочей области.

Для выбора наилучшей структуры компонентов был создан прототип, который тестировался с различными вариантами компоновки элементов: линейное расположение, сетка с фиксированными позициями и свободное размещение с группировкой по темам. Предпочтительным оказался вариант со свободным размещением, позволяющий пользователю гибко организовывать рабочее пространство. Макет интерфейса, использованный для тестирования, представлен на рисунке 6.

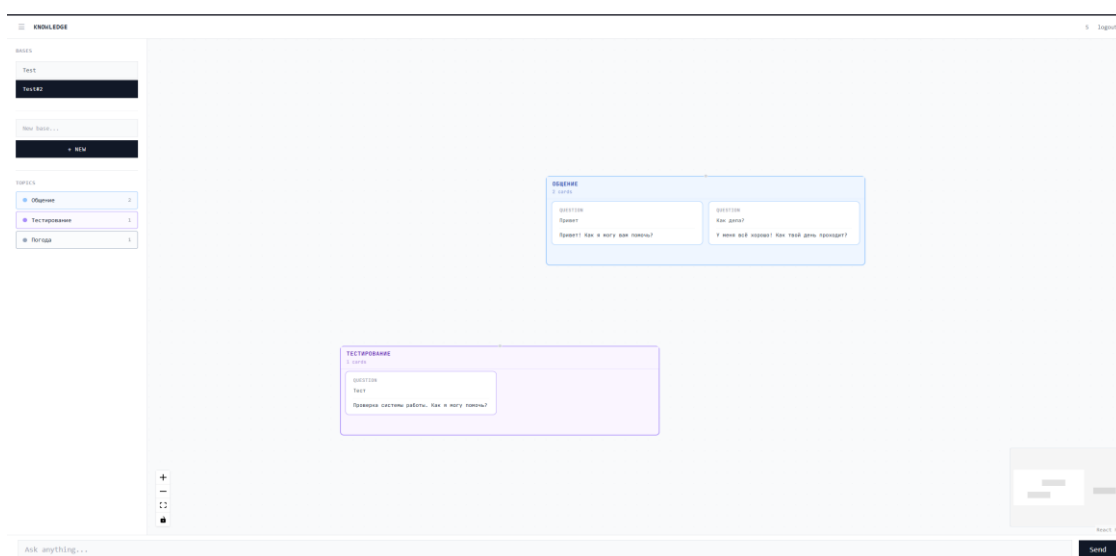


Рисунок 6 – Макет интерфейса приложения

Результаты сравнения подходов к компоновке элементов приведены в таблице 1.5.

Таблица 1.5 – Результаты сравнения подходов к компоновке элементов.

Критерий	Линейное расположение	Фиксированная сетка	Свободное размещение с группировкой
Гибкость размещения	Низкая	Средняя	Высокая
Наглядность при большом количестве элементов	Низкая	Высокая	Высокая
Удобство перемещения элементов	Отсутствует	Ограничено	Полное
Поддержка группировки по темам	Нет	Частично	Да

Исходя из результатов было решено использовать свободное размещение элементов с группировкой по тематическим группам, так как этот подход показал максимальную гибкость и удобство при работе с большим количеством элементов общения с языковой моделью. Для минимизации связанности компонентов был разработан компонент BoardScreen, который инициализируется при выборе диалога и управляет всей логикой интерактивной панели. Дочерние компоненты TopicGroup,

CardNode и ContextMenu отвечают за отрисовку отдельных элементов, что позволяет изменять их поведение независимо друг от друга.

1.2.5 Разработка собственного REST API

В пакете routers были разработаны конечные точки для взаимодействия клиента с сервером. Каждая конечная точка реализована с использованием декораторов FastAPI, определяющих метод HTTP и путь. Для разработки API был выбран подход с разделением логики по модулям-роутерам, что позволяет изолировать функциональность независимых компонентов системы.

Всего было реализовано несколько групп конечных точек, к которым могут обращаться клиентские программы:

- /auth/register и /auth/login - регистрация нового пользователя и вход в систему. Возвращают идентификатор и логин пользователя;
- /boards/ - создание, получение списка и удаление диалогов. Поддерживает фильтрацию по идентификатору пользователя;
- /cards/ - создание карточек, получение всех карточек диалога, обновление темы карточки и удаление;
- /links/ - создание и удаление связей между карточками;
- /ws/chat/{board_id} - WebSocket-соединение для потокового общения с языковой моделью.

Все конечные точки реализованы с использованием декораторов FastAPI, что позволило автоматически сгенерировать интерактивную документацию OpenAPI (Swagger), доступную по адресу /docs. Это упрощает тестирование API и дальнейшее развитие проекта.

Все операции API выполняются над данными, структура которых определяется классами моделей SQLAlchemy. Модели описывают таблицы базы данных и связи между ними, а также содержат методы для выполнения типовых операций выборки и модификации данных. UML-диаграмма классов моделей, с которыми работает API, представлена на рисунке 7.

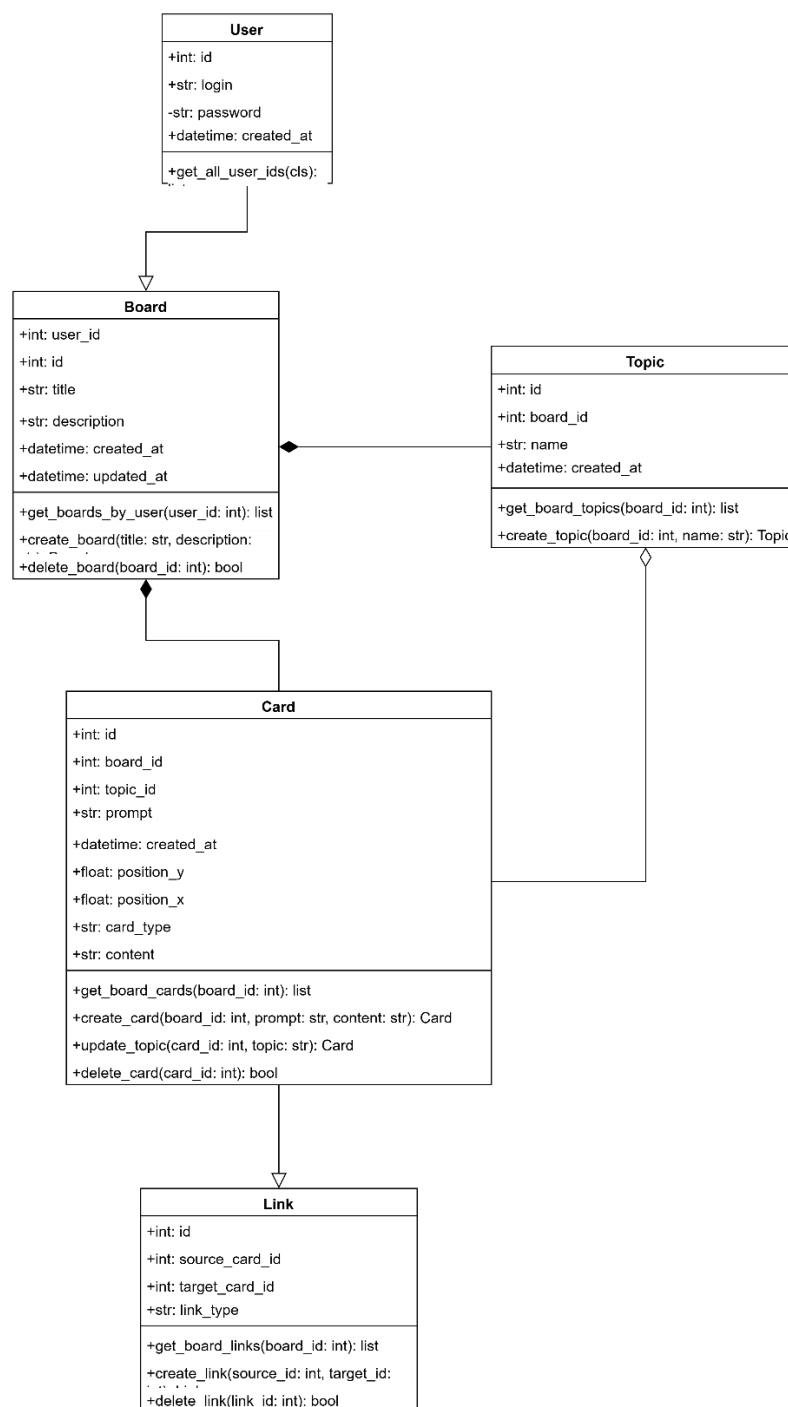


Рисунок 7 – UML-диаграмма классов моделей API

1.2.6 Разработка WebSocket-взаимодействия

Для обеспечения потоковой передачи ответов нейросетевого ассистента в реальном времени было реализовано WebSocket-соединение между клиентом и сервером. В отличие от REST-запросов, которые работают по принципу «запрос-ответ», WebSocket позволяет установить постоянное

двустороннее соединение, по которому сервер может отправлять данные клиенту по мере их появления без необходимости повторных запросов.

На стороне сервера WebSocket-соединение реализовано в модуле chat.py с использованием встроенной поддержки WebSocket фреймворком FastAPI. При подключении клиента создаётся долгоживущее соединение, в рамках которого сервер принимает запросы пользователя, передаёт их языковой модели и отправляет токены ответа клиенту по мере генерации. После завершения генерации ответа соединение остаётся открытым, ожидая следующего запроса.

На стороне клиента WebSocket-соединение устанавливается при выборе диалога. Для каждого диалога создаётся отдельное соединение с уникальным идентификатором. При получении очередного токена от сервера клиент динамически обновляет содержимое карточки на интерактивной панели, что создаёт эффект потоковой генерации текста. При переключении между диалогами предыдущее соединение закрывается и устанавливается новое. Алгоритм WebSocket-взаимодействия представлен на рисунке 8.

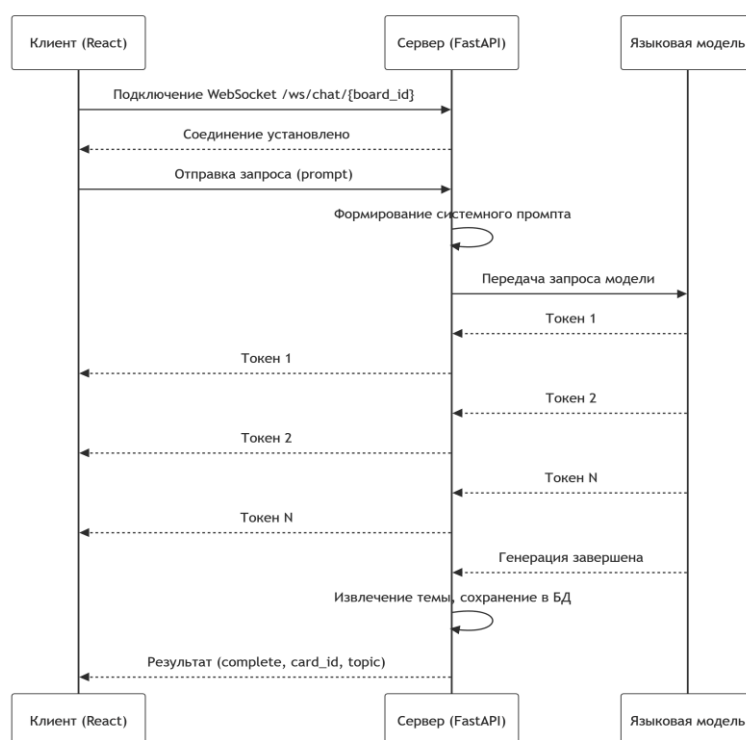


Рисунок 8 – Диаграмма последовательности WebSocket-взаимодействия

1.2.7 Обработка и сохранение контекста диалога

Одной из ключевых задач при построении диалоговой системы с локальной языковой моделью является сохранение контекста общения. В отличие от облачных решений, где модель может обрабатывать сотни тысяч токенов, локальное развёртывание накладывает жёсткие ограничения на размер передаваемых данных. При использовании модели Saiga Gemma 3 на оборудовании с 32 ГБ оперативной памяти размер контекстного окна ограничен 2048 токенами, что не позволяет передавать полную историю диалога с каждым запросом.

Для решения этой проблемы был разработан механизм фактологического контекста. При каждом ответе ассистента из текста ответа выделяется краткая суть — одно-два предложения, содержащие ключевую информацию. В отличие от хранения полного текста ответа, такой подход позволяет сжать информацию диалога до компактного набора фактов, которые сохраняются в базу данных.

При поступлении нового запроса сервер собирает все факты, накопленные за весь диалог, и включает их в системный промпт перед отправкой запроса языковой модели. Это позволяет модели учитывать всё ранее обсуждённое при формировании нового ответа, не перегружая контекстное окно избыточными данными. В результате даже при длительном общении с ассистентом контекст остаётся в пределах допустимого размера, а модель сохраняет понимание общей картины диалога.

Реализация фактологического контекста потребовала расширения схемы базы данных: помимо таблиц для хранения диалогов и сообщений была добавлена таблица для накопленных фактов, каждая запись которой связана с конкретным диалогом. Это позволило вынести значимую информацию за пределы оперативной памяти модели, сохраняя её на уровне сессии. Таким образом, предложенный подход позволяет удерживать контекст длительных бесед в пределах ограниченного окна токенов.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		31

1.3 Анализ результатов

В результате выполнения работы были реализованы серверное и клиентское приложения. Серверная часть написана на языке Python с применением фреймворка FastAPI, клиентская часть — на языке TypeScript с использованием библиотеки React. Для хранения данных используется СУБД SQLite с ORM-библиотекой SQLAlchemy, а взаимодействие с языковой моделью осуществляется через библиотеку llama-cpp-python. Клиентская часть построена на базе библиотеки React Flow, обеспечивающей свободное перемещение элементов на интерактивной панели.

Неавторизованный посетитель имеет минимальный набор возможностей:

- регистрация новой учётной записи в системе;
- вход в систему под существующей учётной записью.

Авторизованный пользователь обладает полным набором возможностей:

- создание нового диалога для общения с нейросетевым ассистентом;
- удаление существующего диалога;
- отправка текстовых запросов к языковой модели;
- получение потоковых ответов в реальном времени;
- просмотр автоматически сформированных тематических групп;
- навигация по тематическим группам через боковую панель;
- свободное перемещение карточек по интерактивной панели;
- перетаскивание карточек между тематическими группами;
- создание связей между карточками;
- удаление отдельных карточек и тематических групп;
- вызов контекстного меню для управления элементами.

Таким образом, разработанное приложение объединяет возможности локальной языковой модели и интерактивной панели для визуальной организации информации.

1.3.1 Тестирование серверной части

Для тестирования и отладки API использовалась автоматически сгенерированная интерактивная документация OpenAPI (Swagger), доступная по адресу /docs. Данный интерфейс позволяет отправлять запросы к любой конечной точке напрямую из браузера, просматривать ответы сервера и проверять корректность работы каждого эндпоинта. На рисунке 9 представлен общий вид страницы документации Swagger с перечнем всех доступных конечных точек.

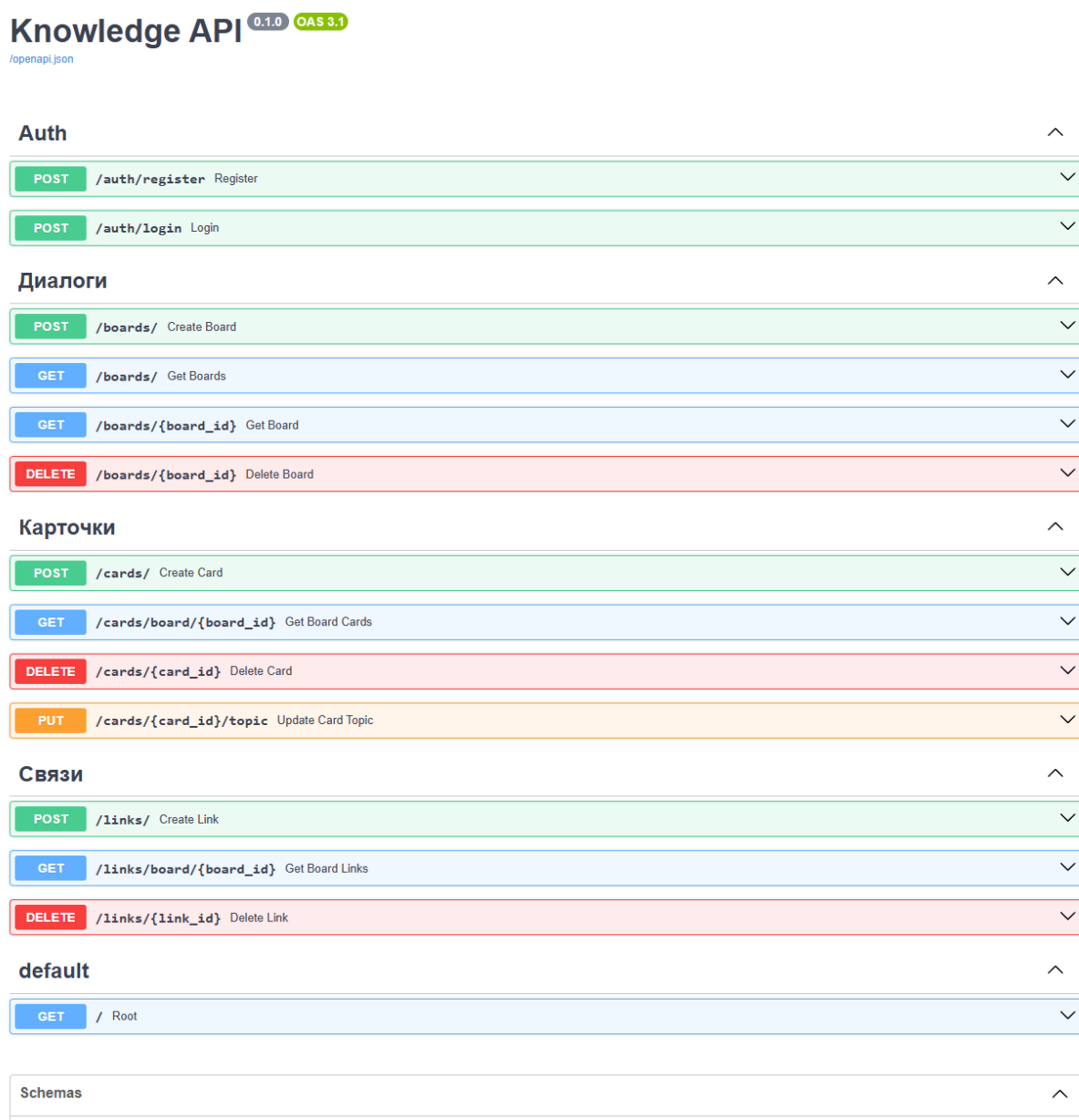


Рисунок 9 – Страница документации Swagger со списком конечных точек

При первом запуске сервера база данных пользователей пуста. Для проверки работоспособности системы авторизации был выполнен запрос на регистрацию нового пользователя через конечную точку /auth/register. На рисунке 10 представлен результат успешной регистрации — сервер вернул идентификатор и логин созданного пользователя.

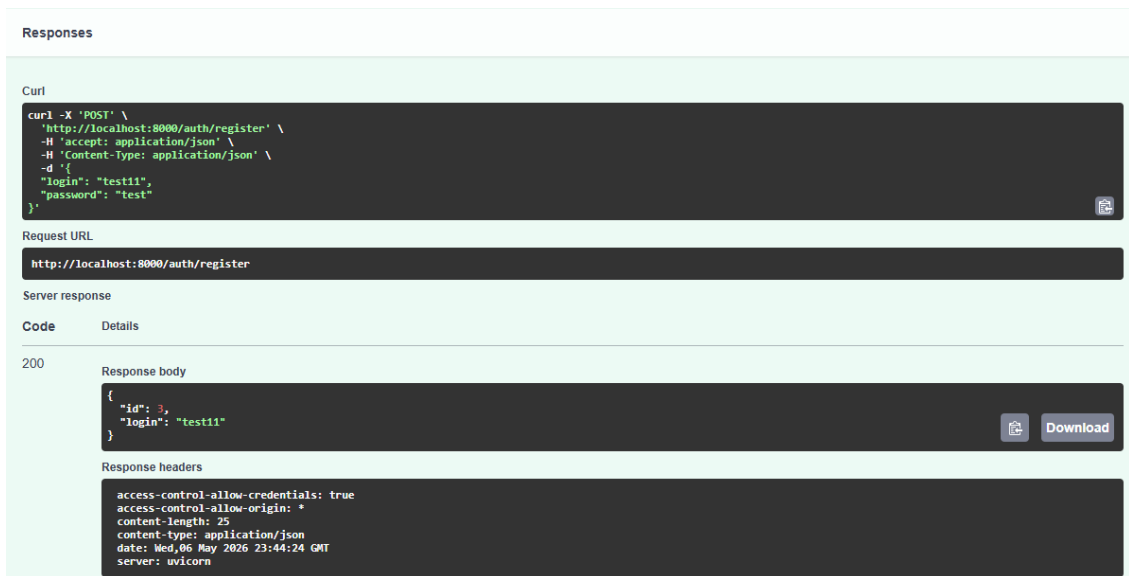


Рисунок 10 – Результат регистрации нового пользователя

После регистрации был выполнен запрос на вход в систему через конечную точку /auth/login. На рисунке 11 представлен успешный вход — сервер вернул данные пользователя, подтверждающие корректность аутентификации.

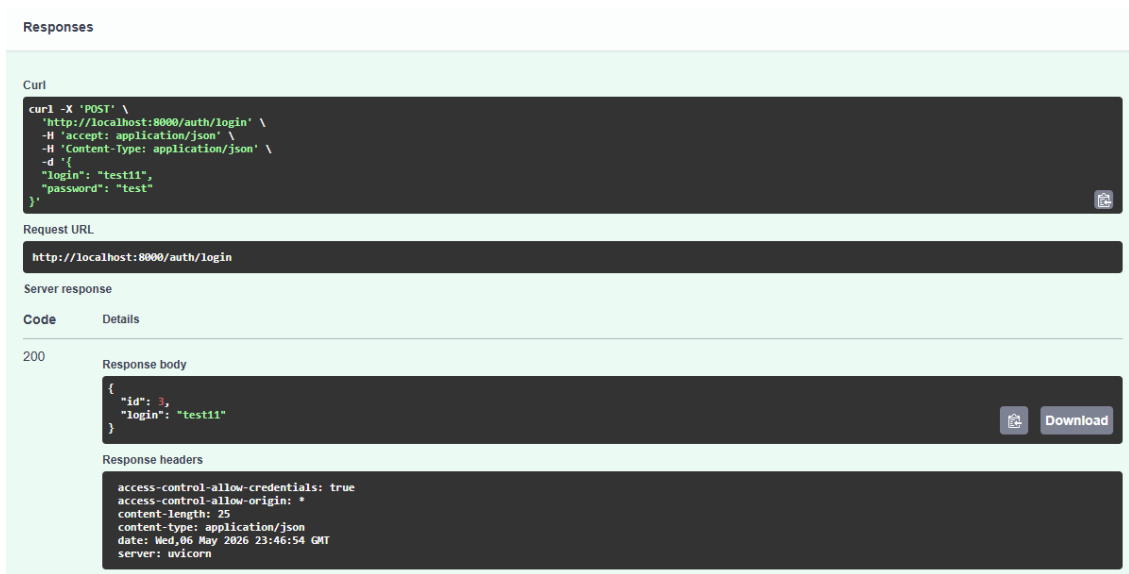


Рисунок 11 – Результат входа в систему

Для проверки работы с диалогами был выполнен запрос на создание нового диалога через конечную точку /boards/. На рисунке 12 представлен результат создания диалога с указанием его идентификатора и заголовка.

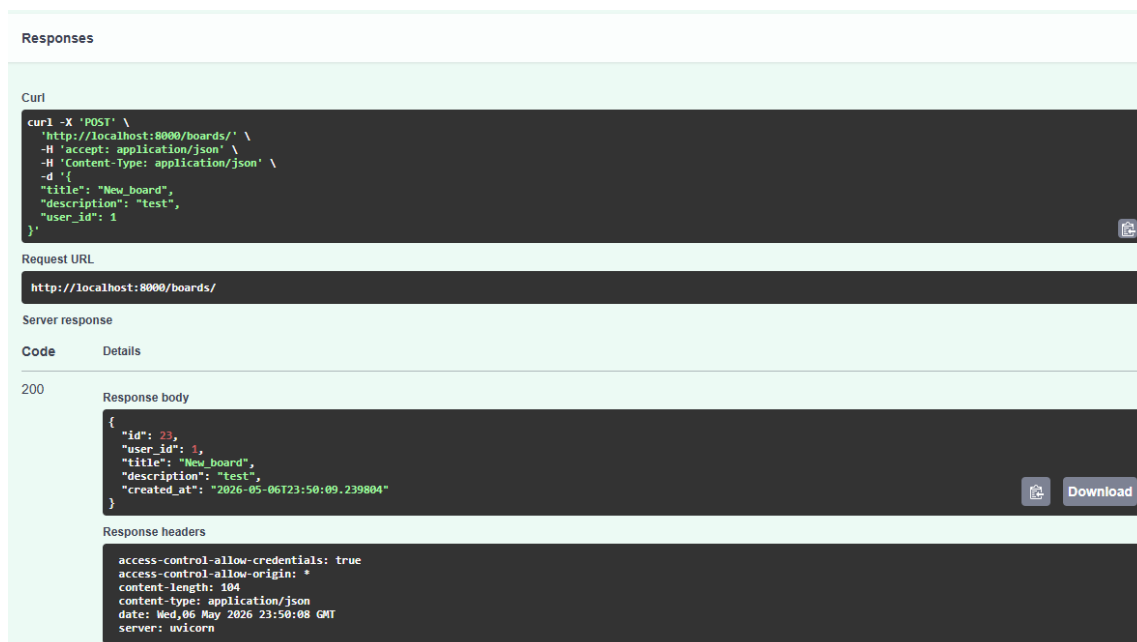


Рисунок 12 – Создание нового диалога

После создания диалога был установлен WebSocket-соединение и отправлен текстовый запрос к нейросетевому ассистенту. На рисунке 13 представлен процесс получения ответа от модели через DevTools — ответ передаётся по токенам в режиме реального времени.

Name	X	Headers	Messages	Initiator	Timing
localhost	⌵	All	▼ Filter using regex (example: (web)?socket)		
css2family=JetBrains+Monowght@400;500;600&display=swap					
main.js?attr=AthN_4Y_xVnq_yHtL3ghMoeY6gGG8d8rIB8KQHdy...			I ["prompt":"проверка связи, скажи что люблю";parent_card_id=null]		67 02:57:31.1
client			I {"type":"chunk","content":""}		30 02:57:31.1
main.tsx			I {"type":"chunk","content": "TOP"}		32 02:57:32.1
tDbv2o=REErYOfZshKuSWU4z3E_BXOPntBRD8yKw8NntkaTog...			I {"type":"chunk","content": "IC"}		31 02:57:32.1
tDbv2o=REErYOfZshKuSWU4z3E_BXOPntBRD8yKw8NntkaTog...			I {"type":"chunk","content": ""}		30 02:57:32.1
@react-refresh			I {"type":"chunk","content": "Ob"}		32 02:57:33.1
react-jsx-dev-runtime.js?v=40a08c22			I {"type":"chunk","content": "ценные"}		34 02:57:33.1
react.js?v=40a08c22			I {"type":"chunk","content": ""}		30 02:57:34.1
react-dom_client.js?v=40a08c22			I {"type":"chunk","content": "Пир"}		33 02:57:34.1
reactflow.js?v=40a08c22			I {"type":"chunk","content": "eer"}		32 02:57:34.1
App.tsx			I {"type":"chunk","content": ""}		30 02:57:35.1
style.css			I {"type":"chunk","content": "Я"}		31 02:57:35.1
index.css			I {"type":"chunk","content": "здесь"}		35 02:57:35.1
env.mjs			I {"type":"chunk","content": ""}		30 02:57:36.1
chunk-573QDKPN.js?v=40a08c22			I {"type":"chunk","content": "чрюбы"}		35 02:57:36.1
chunk-G3PMV6ZZ.js?v=40a08c22			I {"type":"chunk","content": "ответить"}		38 02:57:36.1
chunk-ZSAKTONR.js?v=40a08c22			I {"type":"chunk","content": "на"}		32 02:57:37.1
TopicGroup.tsx			I {"type":"chunk","content": "тео"}		33 02:57:37.1
CardNode.tsx			I {"type":"chunk","content": "х"}		30 02:57:37.1
ThreadEdge.tsx			I {"type":"chunk","content": "вопросы"}		37 02:57:38.1
AuthScreen.tsx			I {"type":"chunk","content": ""}		30 02:57:38.1
ContextMenu.tsx			I {"type":"chunk","content": "Как"}		33 02:57:38.1
axios.js?v=40a08c22			I {"type":"chunk","content": "е"}		31 02:57:39.1
boards?user_id=5			I {"type":"chunk","content": "моя"}		34 02:57:39.1
boards?user_id=5			I {"type":"chunk","content": "помочь"}		36 02:57:39.1
?token=vnbQMjL-7am4F			I {"type":"chunk","content": ""}		30 02:57:40.1
18			I {"type":"complete","card_id":10,"content":"","Привет! Я здесь, чтобы ответить на твои вопросы. Как я могу помочь?","topic":"Общение"}		130 02:57:40.1

Рисунок 13 – Получение ответа от нейросетевого ассистента

1.3.2 Тестирование клиентской части

При запуске клиентского приложения неавторизованный пользователь попадает на экран входа в систему. На рисунке 14 представлено стартовое окно приложения с формой авторизации.

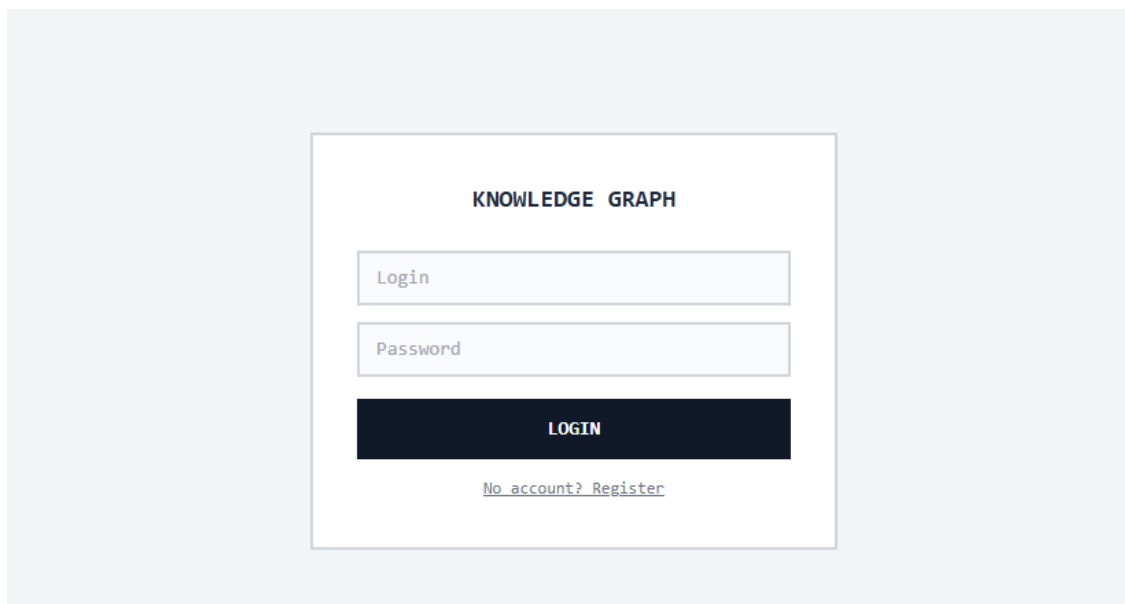


Рисунок 14 – Экран входа в систему

После успешной авторизации пользователь попадает на основной экран приложения. На рисунке 15 представлен основной экран приложения с боковой панелью и пустой рабочей областью.

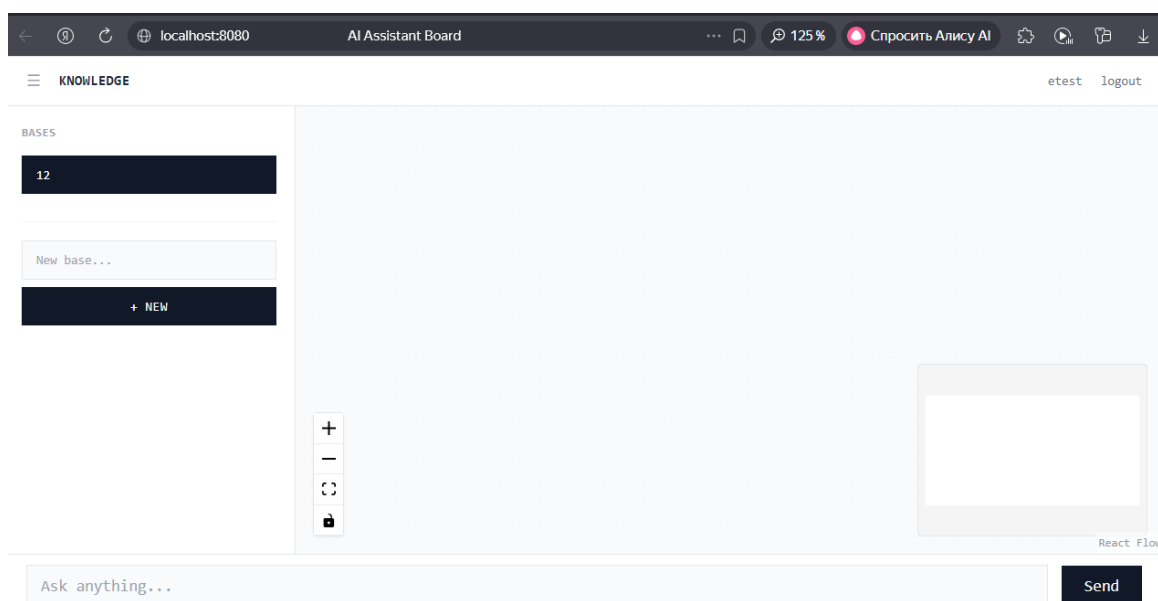


Рисунок 15 – Основной экран приложения

После создания диалога и отправки нескольких запросов к нейросетевому ассистенту на интерактивной панели отображаются тематические группы с карточками. Каждая карточка содержит текст запроса пользователя и ответа ассистента. На рисунке 16 представлен диалог с несколькими тематическими группами на интерактивной панели.

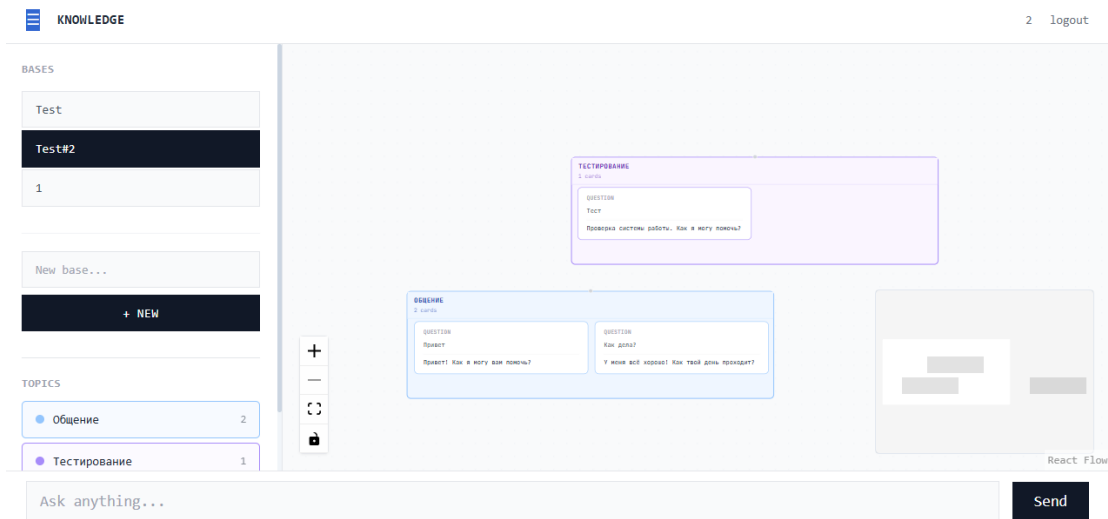


Рисунок 16 – Диалог с тематическими группами на интерактивной панели

Пользователь может перетаскивать карточки между группами. На рисунке 17 представлен результат перетаскивания карточки между тематическими группами.

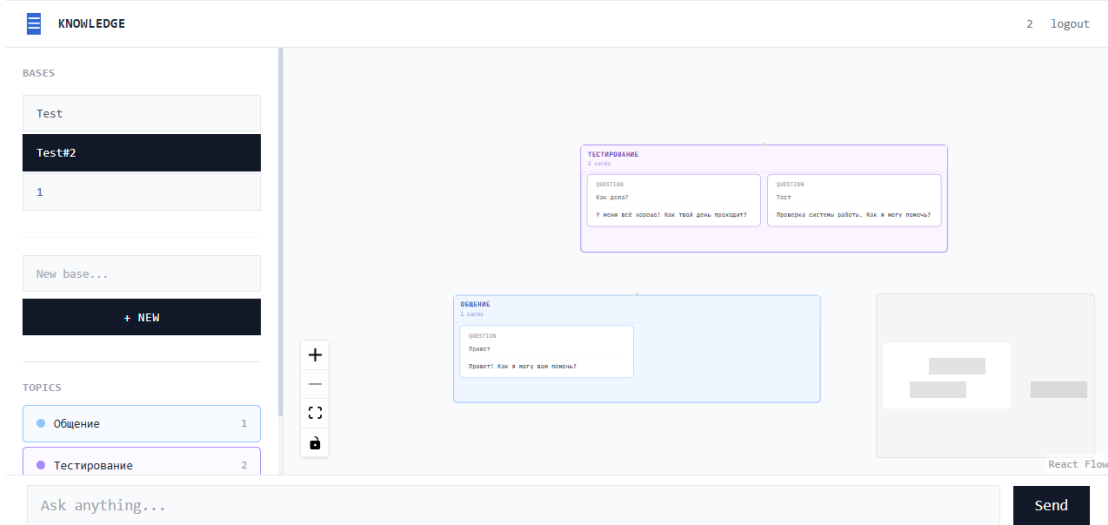


Рисунок 17 – Результат перемещения карточки между тематическими группами

При нажатии правой кнопкой мыши на карточку или группу открывается контекстное меню, позволяющее создать вопрос в выбранной группе, удалить элемент или переместить карточку в другую группу. На рисунке 18 представлено контекстное меню тематической группы

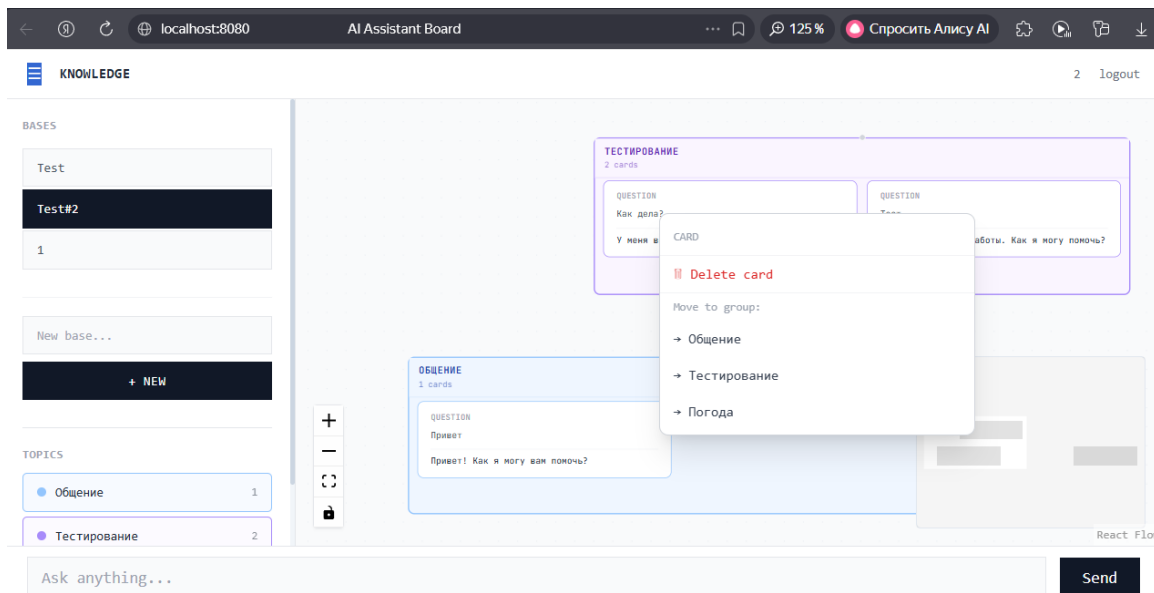


Рисунок 18 – Контекстное меню тематической группы

Клиентское приложение работает корректно, обеспечивает интуитивно понятное взаимодействие с нейросетевым ассистентом и предоставляет пользователю полный набор возможностей для визуальной организации информации на интерактивной панели.

2 Безопасность жизнедеятельности

В данном разделе рассматриваются вопросы обеспечения безопасных условий труда и сохранения здоровья разработчика и пользователей при создании и эксплуатации программного продукта «Нейросетевой ассистент с интерактивной панелью». Процесс разработки веб-приложения связан с длительным применением персонального компьютера, что требует оценки воздействия вредных факторов на организм человека и выработки мер по их минимизации.

2.1 Характеристика вредных факторов при работе с компьютером

Трудовая деятельность разработчика программного обеспечения сопровождается продолжительным нахождением в статичной позе, напряжением зрительных органов и высокой концентрацией внимания. Персональная электронно-вычислительная машина, являясь основным рабочим инструментом, выступает источником целого ряда неблагоприятных воздействий. К числу наиболее значимых вредных факторов относятся электромагнитные поля, создаваемые монитором и системным блоком, длительная зрительная нагрузка при чтении и написании программного кода, статическое напряжение мышц спины и шеи из-за вынужденной рабочей позы, а также повышенный уровень шума от вентиляторов охлаждения. Кроме того, присутствует риск поражения электрическим током при повреждении изоляции проводов или заземляющих элементов оборудования.

Воздействие этих факторов носит накопительный характер и способно привести к развитию профессиональных заболеваний: синдрому сухого глаза, остеохондрозу шейного отдела позвоночника, туннельному синдрому лучезапястного сустава. В связи с этим критически важным становится соблюдение гигиенических норм организации труда и правильная настройка параметров рабочего места.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		39

2.2 Классификация помещения по категориям опасности

2.2.1 Пожарная безопасность

По взрывопожарной и пожарной опасности производственные и офисные помещения классифицируются на пять категорий: А, Б, В1-В4, Г и Д. Рабочий кабинет разработчика, оснащённый исключительно вычислительной техникой, офисной мебелью и бумажными носителями информации, соответствует категории «Д» (пониженная пожароопасность). Данная категория характеризуется тем, что в помещении находятся преимущественно негорючие вещества и материалы в холодном состоянии. Тем не менее, помещение должно быть оснащено первичными средствами пожаротушения, включая углекислотные огнетушители, эффективно устраняющие возгорание электрооборудования без нанесения ему дополнительных повреждений.

2.2.2 Электробезопасность

По опасности поражения человека электрическим током все помещения подразделяются на три класса: без повышенной опасности, с повышенной опасностью и особо опасные. Помещение, в котором выполнялась разработка проекта, относится к категории без повышенной опасности, поскольку в нём отсутствуют токопроводящая пыль, высокая влажность воздуха, металлические полы и химически активная среда.

Для предотвращения поражения электрическим током при эксплуатации персонального компьютера следует соблюдать следующие правила:

- использовать для подключения оборудования только исправные розетки с заземляющим контактом;
- не допускать эксплуатацию устройств с повреждённой изоляцией силовых кабелей;
- исключить расположение проводов питания под напольными покрытиями без дополнительной механической защиты;
- не оставлять включённое оборудование без присмотра на длительное время.

Системный блок и монитор должны быть заземлены. Заземление корпуса компьютера реализуется через трёхпроводную однофазную сеть с заземлённой нейтралью и отдельным защитным проводником. Сопротивление заземляющего устройства для электроустановок напряжением до 1000 В не должно превышать 4 Ом.

2.3 Санитарно-гигиенические требования к организации рабочего места

Санитарные нормы и правила устанавливают количественные требования к параметрам производственной среды, в которой эксплуатируются персональные компьютеры. Их соблюдение позволяет снизить утомляемость и сохранить высокую работоспособность в течение всего трудового дня.

Параметры микроклимата в помещении должны поддерживаться в следующих пределах: температура воздуха 18–21 °С в холодный период года и 22–25 °С в тёплый период, относительная влажность 40–60%, скорость движения воздуха не более 0,1–0,2 м/с. При отклонении от данных значений возрастает риск пересыхания слизистых оболочек глаз, что особенно критично при напряжённой зрительной работе.

Освещённость на поверхности рабочего стола в зоне размещения клавиатуры и документов должна составлять 300–500 люкс. Рекомендуется применять комбинированную систему освещения, при которой естественный свет из окон дополняется локальным светильником с возможностью регулировки яркости и направления светового потока. Экран монитора следует располагать таким образом, чтобы блики от источников света не попадали в поле зрения пользователя.

Площадь, отводимая на одно рабочее место, оснащённое монитором на жидкокристаллической матрице, составляет не менее 4,5 м². Объём помещения на одного сотрудника не может быть менее 15 м³, а высота потолков — менее 3,2 м.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		41

Конструкция рабочего кресла должна предусматривать регулировку высоты сиденья и угла наклона спинки. При правильной посадке стопы полностью опираются на пол, бёдра расположены горизонтально, а предплечья — параллельно поверхности стола. Монитор размещается на расстоянии вытянутой руки пользователя, составляющем 50–70 см. Верхняя граница экрана должна находиться чуть ниже уровня глаз, что обеспечивает угол обзора 15–20 градусов вниз от горизонтальной линии взгляда и способствует меньшему раскрытию глазной щели, уменьшая испарение слёзной плёнки.

2.4 Режим труда и отдыха при работе с ЭВМ

Продолжительность непрерывной работы за компьютером без перерыва не должна превышать одного часа. После каждого часа экранной работы необходим пятнадцатиминутный перерыв, во время которого целесообразно выполнить лёгкую гимнастику для глаз, мышц шеи и плечевого пояса. Упражнения для глаз включают перевод фокуса с ближних объектов на дальние, круговые движения глазами яблоками и частое моргание для увлажнения роговицы. Для профилактики застойных явлений в малом тазу и нижних конечностях рекомендуется раз в два часа вставать с рабочего места и совершать короткую прогулку по помещению.

Суммарная продолжительность непосредственного взаимодействия с экраном монитора не должна превышать шести часов в смену. Оставшееся время распределяется на выполнение организационных задач, обсуждение технических вопросов с коллегами, изучение документации и ведение рабочих записей.

Соблюдение изложенных требований и рекомендаций позволяет минимизировать негативное влияние персонального компьютера на здоровье разработчика и обеспечить безопасные условия труда при создании программного продукта «Нейросетевой ассистент с интерактивной панелью».

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		42

Заключение

В рамках данной работы проведён анализ существующих решений для визуальной организации информации и взаимодействия с нейросетевыми ассистентами. Выявлены ограничения облачных сервисов и профессиональных аналитических пакетов, обоснована целесообразность разработки собственного программного продукта.

Серверная часть реализована на языке Python с использованием фреймворка FastAPI, что позволило обеспечить потоковую передачу данных через WebSocket и автоматическое документирование API. Подключение открытой языковой модели выполнено через библиотеку llama-cpp-python без обращения к сторонним облачным сервисам. Клиентская часть построена на React с применением TypeScript, для визуального управления элементами задействована библиотека React Flow, стилизация выполнена средствами Tailwind CSS. Хранение данных организовано в СУБД SQLite с доступом через SQLAlchemy.

Приложение позволяет создавать и удалять диалоги, отправлять запросы нейросетевой модели с отображением ответа в реальном времени, прерывать генерацию, перемещать и связывать информационные блоки на интерактивной панели, а также сохранять состояние между сеансами. Настроена непрерывная интеграция через GitHub Actions для автоматизированной сборки и доставки обновлений.

Дальнейшее развитие продукта предполагает внедрение авторизации, поддержку нескольких языковых моделей, автоматическое связывание элементов по смысловой близости и экспорт содержимого диалогов в текстовые форматы.

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		43

Список использованных источников

1. Доусон М. Програмируем на Python: основы синтаксиса и практические примеры разработки приложений. — 3-е изд. — СПб.: Прогресс книга, 2019. — 416 с.
2. Бэнкс А., Порселло Е. React: современные шаблоны и архитектурные подходы для разработки клиентских веб-приложений. — СПб.: Питер, 2022. — 320 с.
3. Степанов Д. А. Особенности интеграции открытых языковых моделей в серверные веб-приложения на базе FastAPI // Информационные технологии и вычислительная техника. — 2025. — № 3. — С. 41–47.
4. Официальная документация к асинхронному веб-фреймворку FastAPI. [Электронный ресурс] / Режим доступа: <https://fastapi.tiangolo.com/>, свободный. (дата обращения 12.01.2026).

Приложение А
(обязательное)

Нейросетевой ассистент с интерактивной панелью
UML-Диаграммы

					<i>ПГУ 1.090301.36.001 ПЗ</i>	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

Диаграмма последовательностей

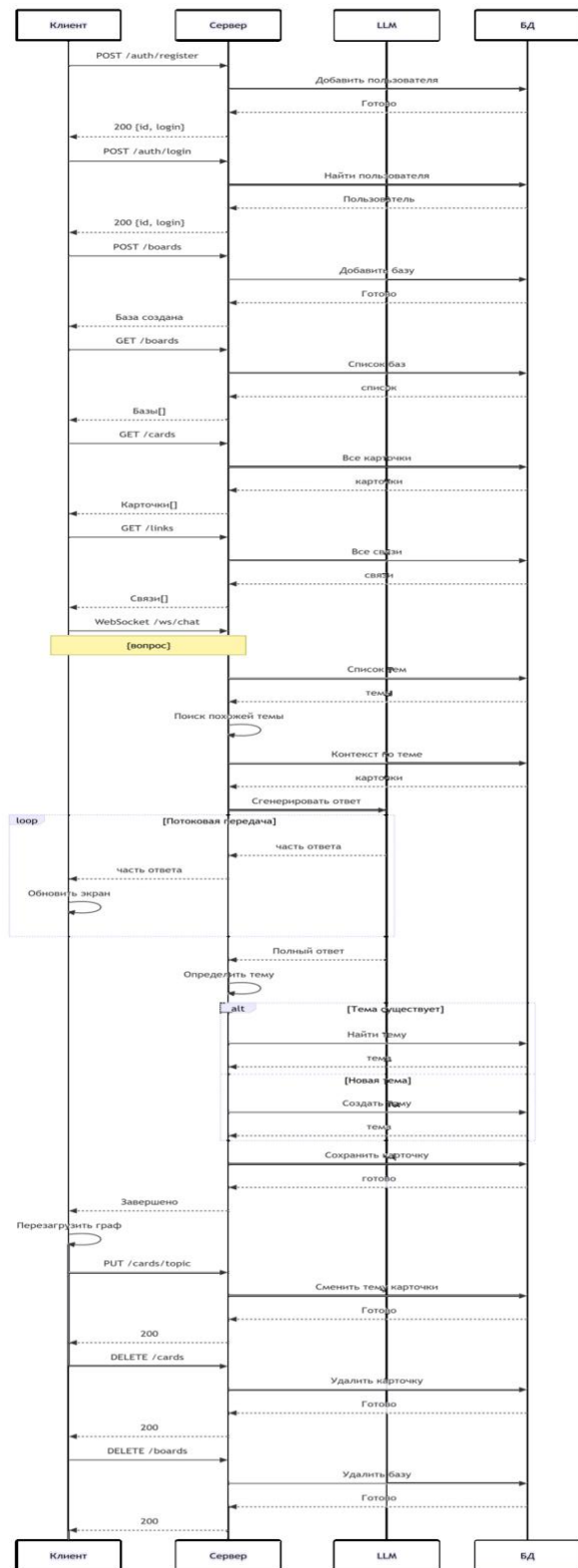
[illegible]

Диаграмма развертывания

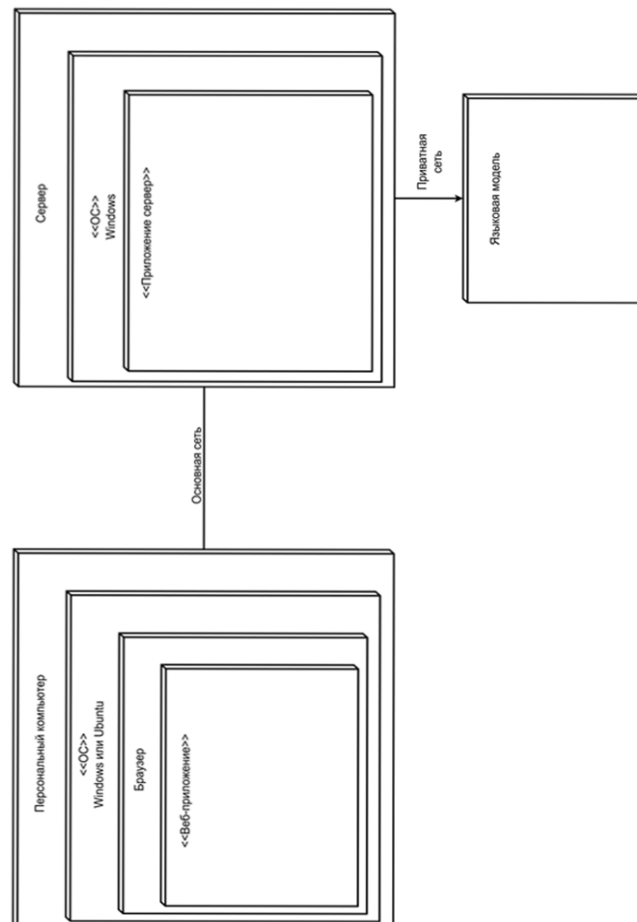
[illegible]

Диаграмма классов

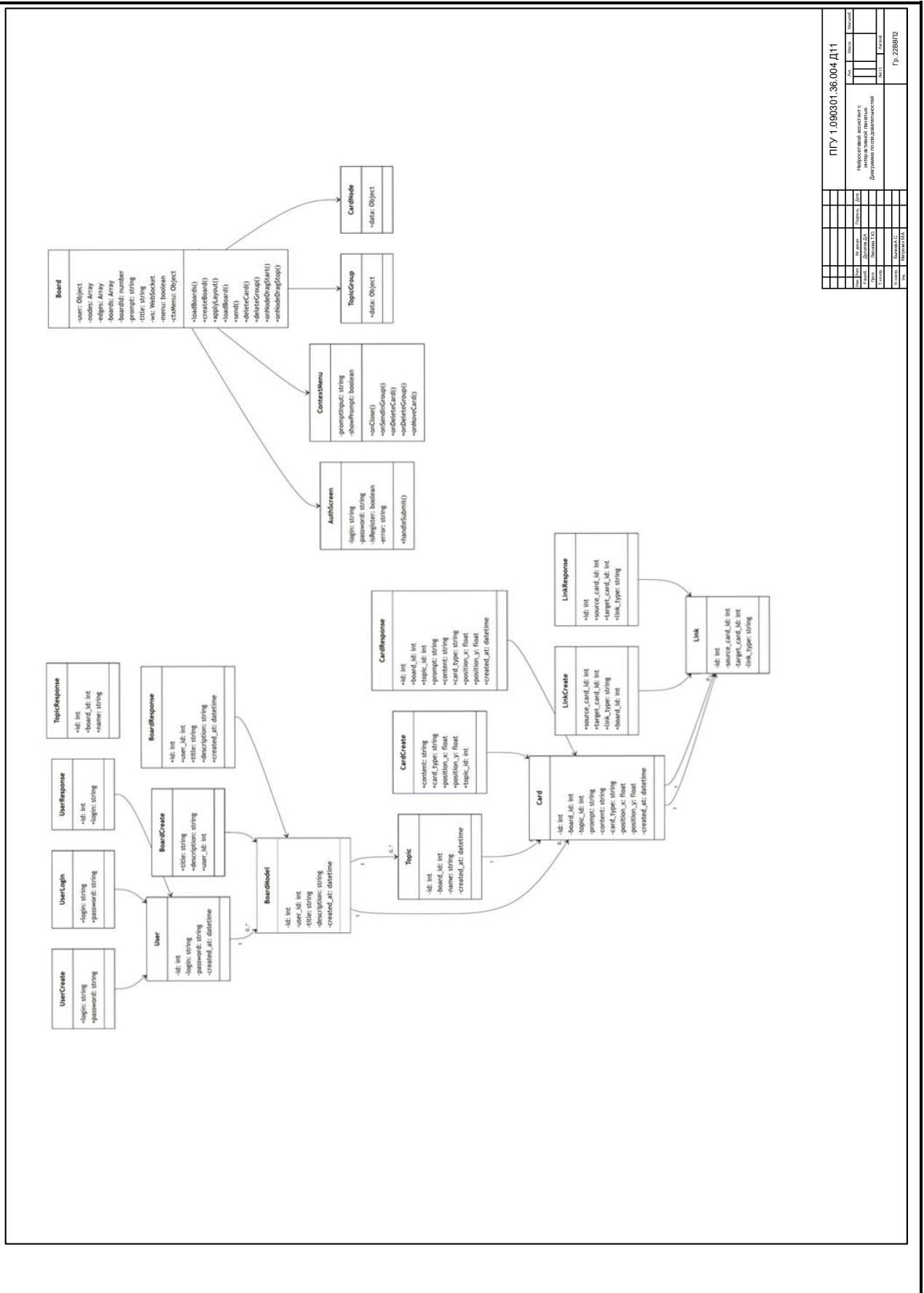
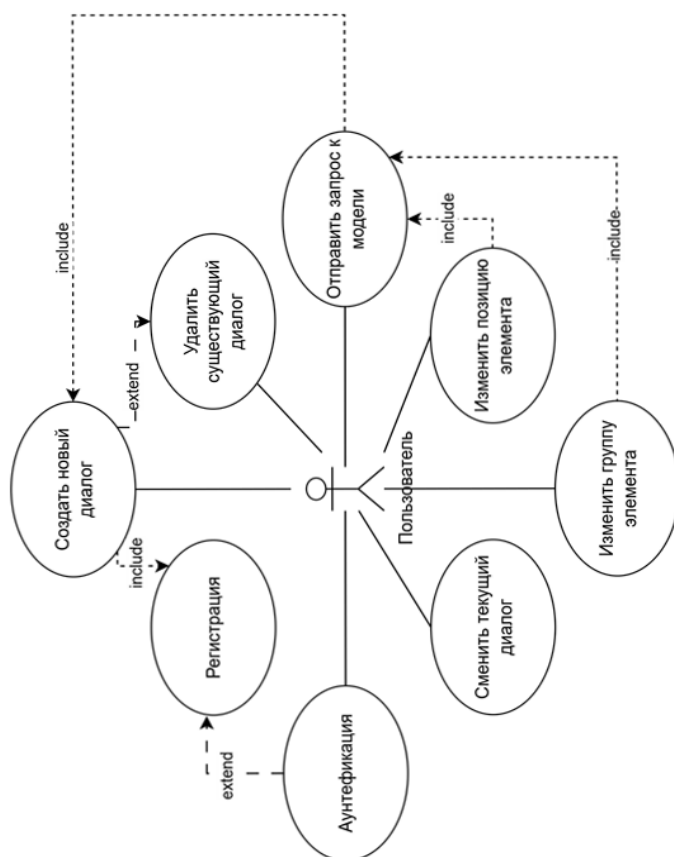


Диаграмма прецедентов

[illegible]

Приложение Б
(обязательное)

Нейросетевой ассистент с интерактивной панелью

Листинг программы

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		50

Файл Schemas.py

```
from pydantic import BaseModel
from typing import Optional, List
from datetime import datetime

class UserCreate(BaseModel):
    login: str
    password: str

class UserLogin(BaseModel):
    login: str
    password: str

class UserResponse(BaseModel):
    id: int
    login: str
    class Config: from_attributes = True

class BoardCreate(BaseModel):
    title: str
    description: Optional[str] = ""
    user_id: Optional[int] = None
    created_at: datetime

class BoardResponse(BaseModel):
    id: int
    user_id: Optional[int] = None
    title: str
    description: Optional[str]
    created_at: datetime
    class Config: from_attributes = True

class CardCreate(BaseModel):
    content: str
    card_type: str = "answer"
    position_x: float = 0
    position_y: float = 0
    topic_id: Optional[int] = None
```

```

class TopicResponse(BaseModel):
    id: int
    board_id: int
    name: str
    class Config: from_attributes = True

class CardCreate(BaseModel):
    content: str
    card_type: str = "answer"
    position_x: float = 0
    position_y: float = 0
    topic_id: Optional[int] = None

class CardResponse(BaseModel):
    id: int
    board_id: int
    topic_id: Optional[int]
    prompt: Optional[str]
    content: str
    card_type: str
    position_x: float
    position_y: float
    created_at: datetime
    class Config: from_attributes = True

class LinkCreate(BaseModel):
    source_card_id: int
    target_card_id: int
    link_type: str = "related"
    board_id: Optional[int] = None

class LinkResponse(BaseModel):
    id: int
    source_card_id: int
    target_card_id: int
    link_type: str
    class Config: from_attributes = True

```

Файл models.py

```
from sqlalchemy import Column, Integer, String, DateTime,
Float, ForeignKey, Text
from sqlalchemy.orm import relationship
from datetime import datetime
from .database import Base

class User(Base):
    __tablename__ = "users"
    id = Column(Integer, primary_key=True, index=True)
    login = Column(String(100), unique=True, nullable=False)
    password = Column(String(200), nullable=False)
    created_at = Column(DateTime, default=datetime.utcnow)
    boards = relationship("Board", back_populates="user",
cascade="all, delete-orphan")

    class Board(Base):
        __tablename__ = "boards"
        id = Column(Integer, primary_key=True, index=True)
        user_id = Column(Integer, ForeignKey("users.id",
ondelete="CASCADE"))
        title = Column(String(200), nullable=False)
        description = Column(Text, nullable=True)
        created_at = Column(DateTime, default=datetime.utcnow)
        user = relationship("User", back_populates="boards")
        cards = relationship("Card", back_populates="board",
cascade="all, delete-orphan")

    class Topic(Base):
        __tablename__ = "topics"
        id = Column(Integer, primary_key=True, index=True)
        board_id = Column(Integer, ForeignKey("boards.id",
ondelete="CASCADE"))
        name = Column(String(100), nullable=False)
        created_at = Column(DateTime, default=datetime.utcnow)
        board = relationship("Board", back_populates="topics")
        cards = relationship("Card", back_populates="topic")
```

```

class Card(Base):
    __tablename__ = "cards"
    id = Column(Integer, primary_key=True, index=True)
    board_id = Column(Integer, ForeignKey("boards.id",
ondelete="CASCADE"))
    topic_id = Column(Integer, ForeignKey("topics.id",
ondelete="SET NULL"), nullable=True)
    prompt = Column(Text, default="")
    content = Column(Text, nullable=False)
    card_type = Column(String(50), nullable=False,
default="answer")
    position_x = Column(Float, default=0)
    position_y = Column(Float, default=0)
    created_at = Column(DateTime, default=datetime.utcnow)
    board = relationship("Board", back_populates="cards")
    topic = relationship("Topic", back_populates="cards")
    outgoing_links = relationship("Link",
foreign_keys="Link.source_card_id", cascade="all, delete-orphan")
    incoming_links = relationship("Link",
foreign_keys="Link.target_card_id", cascade="all, delete-orphan")

class Link(Base):
    __tablename__ = "links"
    id = Column(Integer, primary_key=True, index=True)
    source_card_id = Column(Integer, ForeignKey("cards.id",
ondelete="CASCADE"))

```

Файл main.py

```

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from .database import engine, Base
from .routers import boards, cards, links, chat, auth #
Убедитесь что auth импортирован

Base.metadata.create_all(bind=engine)

```

```

app = FastAPI(title="Knowledge Graph API")
app.add_middleware(CORSMiddleware, allow_origins=["*"],
allow_credentials=True, allow_methods=["*"], allow_headers=["*"])

app.include_router(auth.router)
app.include_router(boards.router)
app.include_router(cards.router)
app.include_router(links.router)
app.include_router(chat.router)
@app.get("/")
def root():
    return {"status": "ok"}

```

Файл database.py

```

from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
import os

# Для разработки используем SQLite (не нужен PostgreSQL)
DATABASE_URL = os.getenv("DATABASE_URL",
"sqlite:///./evidence_board.db")
# Настройка движка
if DATABASE_URL.startswith("sqlite"):
    engine = create_engine(
        DATABASE_URL, connect_args={"check_same_thread":
False}
    )

else:
    engine = create_engine(DATABASE_URL)
    SessionLocal = sessionmaker(autocommit=False,
autoflush=False, bind=engine)
    Base = declarative_base()

```

```
# Зависимость для получения сессии БД
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Файл auth.py

```
from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from ..database import get_db
from ..models import User
from ..schemas import UserCreate, UserLogin, UserResponse
import hashlib

router = APIRouter(prefix="/auth", tags=["Auth"])

def hash_password(password: str) -> str:
    return hashlib.sha256(password.encode()).hexdigest()

@router.post("/register", response_model=UserResponse)
def register(user: UserCreate, db: Session =
Depends(get_db)):
    if db.query(User).filter(User.login ==
user.login).first():
        raise HTTPException(400, "Login already exists")
    db_user = User(login=user.login,
password=hash_password(user.password))
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

@router.post("/login", response_model=UserResponse)
def login(user: UserLogin, db: Session = Depends(get_db)):
    db_user = db.query(User).filter(
        User.login == user.login,
```

```

User.password == hash_password(user.password)
    ).first()
    if not db_user:
        raise HTTPException(401, "Invalid credentials")
    return db_user

```

Файл boards.py

```

from fastapi import APIRouter, Depends, HTTPException, Query
from sqlalchemy.orm import Session
from typing import List, Optional

from .. import schemas, models
from ..database import get_db

router = APIRouter(prefix="/boards", tags=["Доски улик"])

@router.post("/", response_model=schemas.BoardResponse)
def create_board(board: schemas.BoardCreate, db: Session =
Depends(get_db)):
    db_board = models.Board(
        title=board.title,
        description=board.description,
        user_id=board.user_id # Добавлено
    )
    db.add(db_board)
    db.commit()
    db.refresh(db_board)
    return db_board

@router.get("/", response_model=List[schemas.BoardResponse])
def get_boards(
    user_id: Optional[int] = Query(None), # Добавлен параметр
    skip: int = 0,
    limit: int = 100,
    db: Session = Depends(get_db)
):
    query = db.query(models.Board)

```

```

    if user_id is not None:
        query = query.filter(models.Board.user_id == user_id)
    boards = query.offset(skip).limit(limit).all()
    return boards

@router.get("/{board_id}", response_model=schemas.BoardResponse)
def get_board(board_id: int, db: Session = Depends(get_db)):
    board = db.query(models.Board).filter(models.Board.id ==
board_id).first()
    if not board:
        raise HTTPException(status_code=404, detail="Board not
found")
    return board

@router.delete("/{board_id}")
def delete_board(board_id: int, db: Session = Depends(get_db)):
    board = db.query(models.Board).filter(models.Board.id ==
board_id).first()
    if not board:
        raise HTTPException(status_code=404, detail="Board not
found")
    db.delete(board)
    db.commit()
    return {"message": "Board deleted successfully"}

```

Файл cards.py

```

from fastapi import APIRouter, Depends, HTTPException, Query
from sqlalchemy.orm import Session
from typing import List

from .. import schemas, models
from ..database import get_db

router = APIRouter(prefix="/cards", tags=["Карточки"])

@router.post("/", response_model=schemas.CardResponse)
def create_card(
    card: schemas.CardCreate,

```

```

def get_topic_context(db, topic, limit=CONTEXT_LIMIT):
    if not topic:
        return ""
    cards = db.query(models.Card).filter(
        models.Card.topic_id == topic.id
    ).order_by(models.Card.id.desc()).limit(limit).all()
    context = ""
    for card in reversed(cards):
        if card.prompt:
            context += f"Q: {card.prompt}\n"
            context += f"A: {card.content}\n\n"
    return context.strip()

def get_system_prompt(topics, context=None):
    topics_list = ", ".join([t.name for t in topics]) if topics
else "нет тем"
    ctx_block = f"\n\nКонтекст по теме:\n{context}" if context else
    ""
    return f""Ты ассистент Базы Знаний. Отвечай на русском.

Существующие темы: {topics_list}
{ctx_block}

ФОРМАТ ОТВЕТА:
[TOPIC: название_темы] Твой ответ

Если ответ подходит под существующую тему — используй точно такое
же название.

Если тема новая — придумай короткое имя (1-2 слова).
Не используй <think> теги.""

@router.websocket("/ws/chat/{board_id}")
async def websocket_chat(websocket: WebSocket, board_id: int,
db=Depends(get_db)):
    await websocket.accept()
    print(f"WS OPEN board={board_id}")

```

```

try:
    while True:
        data = json.loads(await websocket.receive_text())
        user_prompt = data.get("prompt", "")
        print(f"Q: {user_prompt}")

        # Поиск подходящей темы через эмбединги
        best_topic      =      find_best_topic(db,      board_id,
user_prompt)

        context  =  get_topic_context(db,  best_topic)  if
best_topic else None
        print(f"BEST TOPIC: {best_topic.name if best_topic else
'NEW'}, SCORE: {best_topic.name if best_topic else 'N/A'}")

        all_topics                                     =
db.query(models.Topic).filter(models.Topic.board_id == board_id).all()
        system_prompt = get_system_prompt(all_topics, context)

        formatted                                     =
f"<bos><start_of_turn>system\n{system_prompt}<end_of_turn>\n<start_of_
turn>user\n{user_prompt}<end_of_turn>\n<start_of_turn>assistant\n"

        full = ""
        stream = llm(
            formatted,
            max_tokens=500,
            temperature=0.6,
            top_p=0.9,
            stop=["<end_of_turn>", "<start_of_turn>"],
            echo=False,
            stream=True
        )
        for chunk in stream:
            if "choices" in chunk:
                t = chunk["choices"][0].get("text", "")
                if t:

```

```

        full += t
        if '<think>' not in t and '</think>' not in
t:
            await websocket.send_json({"type":
"chunk", "content": t})
            await asyncio.sleep(0.01)

    print(f"RAW: '{full[:300]}'")

    full = re.sub(r'<think>.*?</think>', '', full,
flags=re.DOTALL).strip()
    if not full:
        full = re.sub(r'<[^>]+>', '', full).strip()

    match = re.search(r'\[(?:TOPIC|GENERAL):\s*(.+?)\]',
full)
    topic_name = match.group(1).strip() if match else
"general"

    clean = re.sub(r'\[(?:TOPIC|GENERAL):\s*.*?\]\s*', '',
full).strip()

    if not clean:
        clean = full

    print(f"A: [{topic_name}] {clean[:100]}")

    topic = db.query(models.Topic).filter(
        models.Topic.board_id == board_id,
        models.Topic.name == topic_name
    ).first()
    if not topic:
        topic = models.Topic(board_id=board_id,
name=topic_name)

        db.add(topic)
        db.commit()
        db.refresh(topic)

```

```

card = models.Card(
    board_id=board_id, topic_id=topic.id,
    prompt=user_prompt, content=clean,
    card_type=topic_name, position_x=100,
position_y=100
)
db.add(card)
db.commit()
db.refresh(card)

await websocket.send_json({
    "type": "complete",
    "card_id": card.id,
    "content": clean,
    "topic": topic_name
})

except WebSocketDisconnect:
    print("WS CLOSE")
except Exception as e:
    print(f"ERROR: {e}")
    import traceback
    traceback.print_exc()

```

Файл links.py

```

from fastapi import APIRouter, Depends, HTTPException
from sqlalchemy.orm import Session
from typing import List

from .. import schemas, models
from ..database import get_db

router = APIRouter(prefix="/links", tags=["Связи"])

@router.post("/", response_model=schemas.LinkResponse)
def create_link(link: schemas.LinkCreate, db: Session =
Depends(get_db)):

```

```

        source = db.query(models.Card).filter(models.Card.id ==
link.source_card_id).first()
        target = db.query(models.Card).filter(models.Card.id ==
link.target_card_id).first()

    if not source or not target:
        raise HTTPException(status_code=404, detail="Card not
found")

    db_link = models.Link(
        source_card_id=link.source_card_id,
        target_card_id=link.target_card_id,
        link_type=link.link_type
    )
    db.add(db_link)
    db.commit()
    db.refresh(db_link)
    return db_link

@router.get("/board/{board_id}",
response_model=List[schemas.LinkResponse])
def get_board_links(board_id: int, db: Session = Depends(get_db)):
    links = db.query(models.Link).join(
        models.Card, models.Link.source_card_id == models.Card.id
    ).filter(models.Card.board_id == board_id).all()
    return links

@router.delete("/{link_id}")
def delete_link(link_id: int, db: Session = Depends(get_db)):
    link = db.query(models.Link).filter(models.Link.id ==
link_id).first()
    if not link:
        raise HTTPException(status_code=404, detail="Link not
found")

    db.delete(link)
    db.commit()

```

Файл App.tsx

```
import { useState, useCallback, useEffect, useRef } from 'react';
import ReactFlow, {
  Node, Edge, useEdgesState,
  Background, Controls, MiniMap, MarkerType,
  applyNodeChanges, useReactFlow,
} from 'reactflow';
import 'reactflow/dist/style.css';
import { TopicGroup } from '../components/TopicGroup';
import { CardNode } from '../components/CardNode';
import { ThreadEdge } from '../components/ThreadEdge';
import { AuthScreen } from '../components/AuthScreen';
import { ContextMenu } from '../components/ContextMenu';
import axios from 'axios';

const nodeTypes = { topicGroup: TopicGroup, cardNode: CardNode };
const edgeTypes = { thread: ThreadEdge };

const TOPIC_COLORS = [
  { bg: '#f8fafc', border: '#94a3b8', text: '#475569' },
  { bg: '#faf5ff', border: '#a78bfa', text: '#5b21b6' },
  { bg: '#f0fdf4', border: '#86efac', text: '#166534' },
  { bg: '#eff6ff', border: '#93c5fd', text: '#1e40af' },
  { bg: '#fff7ed', border: '#fdb762', text: '#9a3412' },
];

const CARD_W = 320;
const CARD_H = 130;
const GAP = 12;
const HEADER_H = 44;
const COLS = 2;

function getTopicColor(name: string) {
  const h = name.split('').reduce((a, c) => a + c.charCodeAt(0), 0);
  return TOPIC_COLORS[h % TOPIC_COLORS.length];
}
```

```

    }

    function groupWidth() { return CARD_W * COLS + GAP * (COLS + 1); }
    function groupHeight(count: number) {
        const rows = Math.ceil(Math.max(1, count) / COLS);
        return HEADER_H + rows * CARD_H + GAP * (rows + 1);
    }

    function cardAbsPos(groupPos: {x: number, y: number}, index:
number) {
        const col = index % COLS;
        const row = Math.floor(index / COLS);
        return {
            x: groupPos.x + GAP + col * (CARD_W + GAP),
            y: groupPos.y + HEADER_H + GAP + row * (CARD_H + GAP),
        };
    }

    function getFreePosition(nodes: Node[]) {
        const used = nodes.filter(n => n.type === 'topicGroup').map(n =>
n.position);
        const gw = groupWidth();
        for (let y = 80; y < 3000; y += 350) {
            for (let x = 30; x < 2000; x += gw + 50) {
                if (!used.some(p => Math.abs(p.x - x) < gw + 30 &&
Math.abs(p.y - y) < 320)) {
                    return { x, y };
                }
            }
        }
        return { x: 30, y: used.length * 350 + 80 };
    }

    function App() {
        const [user, setUser] = useState<{ id: number; login: string } |
null>(null);
        const [ready, setReady] = useState(false);

        useEffect(() => {

```

```

const s = localStorage.getItem('user');
    if (s) try { setUser(JSON.parse(s)); } catch {
localStorage.removeItem('user'); }
    setReady(true);
}, []);

if (!ready) return <div className="h-screen flex items-center
justify-center bg-gray-50 font-mono text-gray-400">...</div>;
    if (!user) return <AuthScreen onLogin={u => { setUser(u);
localStorage.setItem('user', JSON.stringify(u)); }} />;
    return <Board user={user} onLogout={() => { setUser(null);
localStorage.removeItem('user'); }} />;
}

function Board({ user, onLogout }: { user: { id: number; login:
string }; onLogout: () => void }) {
    const [nodes, setNodes] = useState<Node[]>([]);
    const [edges, setEdges, onEdgesChange] = useEdgesState([]);
    const [boards, setBoards] = useState<any[]>([]);
    const [boardId, setBoardId] = useState<number | null>(null);
    const [prompt, setPrompt] = useState('');
    const [title, setTitle] = useState('');
    const [ws, setWs] = useState<WebSocket | null>(null);
    const streamRef = useRef<string | null>(null);
    const [menu, setMenu] = useState(false);
    const [ctxMenu, setCtxMenu] = useState<{ x: number; y: number;
nodeId: string; type: string } | null>(null);
    const dragRef = useRef<{ nodeId: string; sourceGroup: string |
null } | null>(null);
    const reactFlow = useReactFlow();

    const loadBoards = useCallback(async () => {
        const { data } = await axios.get(`/boards?user_id=${user.id}`);
        setBoards(data);
    }, [user.id]);

    useEffect(() => { loadBoards(); }, [loadBoards]);
    const createBoard = async () => {

```

```

    if (!title.trim()) return;
    await axios.post('/boards/', { title, description: '', user_id:
user.id });
    setTitle('');
    loadBoards();
  };
  const applyLayout = useCallback((nds: Node[]) => {
    return nds.map(node => {
      if (node.type === 'topicGroup') {
        const count = nds.filter(n => (n as any).groupId ===
node.id).length;
        return {
          ...node,
          style: { ...node.style, width: groupWidth(), height:
groupHeight(count) },
          data: { ...node.data, count },
        };
      }
      if (node.type === 'cardNode' && (node as any).groupId) {
        const group = nds.find(n => n.id === (node as any).groupId);
        if (group) {
          const cards = nds.filter(n => (n as any).groupId ===
group.id);
          const idx = cards.indexOf(node);
          if (idx >= 0) return { ...node, position:
cardAbsPos(group.position, idx) };
        }
      }
      return node;
    });
  }, []);

  const onNodesChangeCustom = useCallback((changes: any[]) => {
    setNodes(prev => {
      const movedGroups = changes
        .filter(c => c.type === 'position' && c.dragging &&
c.position)

```

					ПГУ 1.090301.36.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

.map(c => prev.find(n => n.id === c.id))
    .filter(n => n?.type === 'topicGroup');

let updated = applyNodeChanges(changes, prev);

if (movedGroups.length > 0) {
    updated = updated.map(node => {
        if (node.type === 'cardNode' && (node as any).groupId) {
            const group = movedGroups.find(g => g!.id === (node as
any).groupId) || updated.find(n => n.id === (node as any).groupId);
            if (group) {
                const cards = updated.filter(n => (n as any).groupId
=== group.id);

                const idx = cards.indexOf(node);
                if (idx >= 0) return { ...node, position:
cardAbsPos(group.position, idx) };
            }
            return node;
        }
    });
    return updated;
}

const focusGroup = useCallback((groupId: string) => {
    const group = nodes.find(n => n.id === groupId);
    if (group) {
        reactFlow.setCenter(group.position.x + groupWidth() / 2,
group.position.y + 200, { zoom: 1, duration: 500 });
    }
}, [nodes, reactFlow]);

const loadBoard = useCallback(async () => {
    if (!boardId) return;
    const [{ data: cards }, { data: links }] = await Promise.all([
        axios.get(`/cards/board/${boardId}`),
        axios.get(`/links/board/${boardId}`)
    ]);

```

```

]);

const groups: Record<string, any[]> = {};
cards.forEach((c: any) => {
    const t = c.card_type || 'general';
    if (!groups[t]) groups[t] = [];
    groups[t].push(c);
});

let rawNodes: any[] = [];
const es: Edge[] = [];
const topics = Object.keys(groups);

topics.forEach((topic) => {
    const color = getTopicColor(topic);
    const topicCards = groups[topic];
    const pos = getFreePosition(rawNodes);
    const gId = `g-${topic}`;

    rawNodes.push({
        id: gId, type: 'topicGroup', position: pos,
        style: { width: groupWidth(), height:
groupHeight(topicCards.length) },
        data: { label: topic, count: topicCards.length, color:
color.border, bg: color.bg, text: color.text },
    });

    topicCards.forEach((card: any, ci: number) => {
        rawNodes.push({
            id: String(card.id), type: 'cardNode',
            position: cardAbsPos(pos, ci),
            data: { prompt: card.prompt, content: card.content,
color: color.border, cardId: card.id, topic },
            groupId: gId,
        });
    });
});
});

```

```

links.forEach((l: any) => {
    es.push({
        id: `l-${l.id}`, source: String(l.source_card_id), target:
String(l.target_card_id),
        type: 'thread', style: { stroke: '#cbd5e1', strokeWidth: 1
},
        markerEnd: { type: MarkerType.ArrowClosed, color:
'#cbd5e1', width: 5, height: 5 },
    });
});

setNodes(rawNodes);
setEdges(es);
}, [boardId]);

useEffect(() => { if (boardId) loadBoard(); }, [boardId,
loadBoard]);

useEffect(() => {
    if (!boardId) return;
    const s = new WebSocket(`${location.protocol === 'https:' ?
'wss:' : 'ws:'}://${location.host}/ws/chat/${boardId}`);
    s.onmessage = (e) => {
        const d = JSON.parse(e.data);
        if (d.type === 'chunk' && streamRef.current) {
            setNodes(prev => prev.map(n => n.id === streamRef.current
? { ...n, data: { ...n.data, content: n.data.content + d.content } } :
n));
        }
        if (d.type === 'complete') {
            streamRef.current = null;
            setTimeout(loadBoard, 300);
        }
    };
});

setWs(s);

```

```

        return () => s.close();
    }, [boardId]);
const deleteCard = async (cardId: string) => {
    await axios.delete(`/cards/${cardId}`).catch(()=>{});
    loadBoard();
};

const deleteGroup = async (groupId: string) => {
    for (const c of nodes.filter((n: any) => n.groupId === groupId))
    {
        if (!c.id.startsWith('t-')) await
        axios.delete(`/cards/${(c.data as any).cardId || c.id}`).catch(()=>{});
    }
    loadBoard();
};

const onNodeDragStart = useCallback((_e: React.MouseEvent, node:
Node) => {
    if (node.type === 'cardNode') dragRef.current = { nodeId:
node.id, sourceGroup: (node as any).groupId || null };
    }, []);

const onNodeDragStop = useCallback((_e: React.MouseEvent, node:
Node) => {
    if (node.type !== 'cardNode' || !dragRef.current) return;
    const cx = node.position.x + CARD_W / 2;
    const cy = node.position.y + CARD_H / 2;
    let target: string | null = null;
    for (const g of nodes.filter(n => n.type === 'topicGroup')) {
        const gw = (g.style as any)?.width || groupWidth();
        const gh = (g.style as any)?.height || 200;
        if (cx > g.position.x && cx < g.position.x + gw && cy >
g.position.y && cy < g.position.y + gh) { target = g.id; break; }
    }
    setNodes(prev => {
        let upd = prev.map(n => {

```

```

    if (n.id === node.id) {
        const obj: any = { ...n };
        if (target) { obj.groupId = target; obj.data = {
...obj.data, topic: target.replace('g-', '') }; }
        return obj;
    }
    return n;
});
return applyLayout(upd);
});
if (target && target !== dragRef.current?.sourceGroup)
moveCardToGroup(node.id, target);
dragRef.current = null;
}, [nodes, applyLayout]);
const onNodeContextMenu = useCallback((e: React.MouseEvent, node:
Node) => {
    e.preventDefault();
    setCtxMenu({ x: e.clientX, y: e.clientY, nodeId: node.id, type:
node.type || 'cardNode' });
}, []);

const topicGroups = nodes.filter(n => n.type === 'topicGroup');

return (
    <div className="h-screen flex flex-col bg-gray-50 font-mono
text-gray-800" onClick={() => setCtxMenu(null)}>
        <header className="h-11 bg-white border-b border-gray-200
flex items-center justify-between px-5 shrink-0">
            <div className="flex items-center gap-4">
                <button onClick={() => setMenu(!menu)} className="text-
gray-400 hover:text-gray-700 text-sm">☰</button>
                <span className="text-xs font-semibold tracking-wide
uppercase">Knowledge</span>
            </div>
            <div className="flex items-center gap-4 text-[11px] text-
gray-500">
                <span>{user.login}</span>

```

```

        <button      onClick={onLogout}      className="hover:text-gray-
800">logout</button>
      </div>
    </header>

    <div className="flex-1 flex overflow-hidden">
      {menu && (
        <div className="w-64 bg-white border-r border-gray-200
overflow-y-auto shrink-0 p-4">
          <h3 className="text-[10px] font-bold uppercase text-
gray-400 mb-3 tracking-wider">Bases</h3>
          <div className="space-y-0.5 mb-6">
            {boards.map(b => (
              <button      key={b.id}      onClick={() =>
setBoardId(b.id)}
              className={`block w-full text-left px-3 py-2
text-[11px] border transition-colors ${boardId === b.id ? 'bg-gray-900
text-white border-gray-900' : 'bg-gray-50 border-gray-200 text-gray-600
hover:bg-gray-100'}`}>
                {b.title}
              </button>
            ))}
          </div>
          <div className="pt-4 border-t border-gray-100">
            <input      value={title}      onChange={e =>
setTitle(e.target.value)}
            placeholder="New base..." className="w-full px-3
py-2 text-[11px] border border-gray-200 focus:outline-none focus:border-
gray-400 bg-gray-50"
            onKeyDown={e => e.key === 'Enter' && createBoard()}
          />
            <button onClick={createBoard}
            className="w-full mt-1.5 px-3 py-2 text-[11px]
font-medium bg-gray-900 text-white border border-gray-900 hover:bg-gray-
800 uppercase tracking-wider">+ New</button>
          </div>

```

```

    { /* Панель "Темы диалогов" */ }
    { boardId && topicGroups.length > 0 && (
      <div className="mt-6 pt-4 border-t border-gray-200">
        <h3 className="text-[10px] font-bold uppercase
text-gray-400 mb-2 tracking-wider">Topics</h3>
        <div className="space-y-1">
          { topicGroups.map(g => (
            <button key={g.id} onClick={() =>
focusGroup(g.id)}
              className="w-full text-left px-3 py-2 text-
[11px] rounded border transition-colors hover:bg-gray-100 flex items-
center gap-2"
              style={{ borderColor: (g.data as any).color,
backgroundColor: (g.data as any).bg + '80' }} >
                <span className="w-2 h-2 rounded-full"
style={{ backgroundColor: (g.data as any).color }} />
                <span className="flex-1 truncate">{(g.data
as any).label}</span>
                <span className="text-[10px] opacity-
50">{(g.data as any).count}</span>
              </button>
            ) ) }
          </div>
        </div>
      ) }
    </div>
    <div className="flex-1">
      { !boardId ? (
        <div className="h-full flex items-center justify-
center text-gray-400 text-sm">Select a base to start</div>
      ) : (
        <ReactFlow nodes={nodes} edges={edges}
          onNodesChange={onNodesChangeCustom}
          onEdgesChange={onEdgesChange}
          nodeTypes={nodeTypes} edgeTypes={edgeTypes} fitView
          onNodeDragStart={onNodeDragStart}

```

Файл index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

body { font-family: 'JetBrains Mono', 'Fira Code', monospace;
background: #f8fafc; color: #1e293b; }

::-webkit-scrollbar { width: 4px; }
::-webkit-scrollbar-track { background: #f1f5f9; }
::-webkit-scrollbar-thumb { background: #cbd5e1; border-radius:
2px; }

.react-flow__node { font-family: 'JetBrains Mono', monospace
!important; }

/* Группы всегда под карточками */
.react-flow__node-topicGroup { z-index: 0 !important; }
.react-flow__node-cardNode { z-index: 10 !important; }
.react-flow__node-aiNode { z-index: 20 !important; }
.react-flow__node-cardNode.selected { z-index: 100 !important; }
```

Файл main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import { ReactFlowProvider } from 'reactflow'
import App from './App'
import 'reactflow/dist/style.css'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <ReactFlowProvider>
      <App />
    </ReactFlowProvider>
  </React.StrictMode>,)
```

Файл AuthScreen.tsx

```
import { useState } from 'react';
import axios from 'axios';

interface Props {
  onLogin: (user: { id: number; login: string }) => void;
}

export function AuthScreen({ onLogin }: Props) {
  const [login, setLogin] = useState('');
  const [password, setPassword] = useState('');
  const [isRegister, setIsRegister] = useState(false);
  const [error, setError] = useState('');
  const handleSubmit = async () => {
    if (!login.trim() || !password.trim()) { setError('Fill all fields'); return; }
    try {
      const endpoint = isRegister ? '/auth/register' : '/auth/login';
      const res = await axios.post(endpoint, { login: login.trim(), password });
      onLogin(res.data);
    } catch (err: any) {
      setError(err.response?.data?.detail || 'Error');
    }
  };
  return (
    <div className="h-screen flex items-center justify-center bg-gray-100 font-mono">
      <div className="bg-white border-2 border-gray-300 p-8 w-full max-w-sm">
        <h1 className="text-lg font-bold text-center mb-6 uppercase">Knowledge Graph</h1>
        <input type="text" value={login} onChange={(e) => setLogin(e.target.value)}
          placeholder="Login" className="w-full px-3 py-2 border-2 border-
```

```

    gray-300 mb-3 text-sm focus:outline-none focus:border-gray-900 bg-
gray-50"

    onKeyDown={(e) => e.key === 'Enter' && handleSubmit()} />
    <input type="password" value={password} onChange={(e) =>
setPassword(e.target.value)}

    placeholder="Password" className="w-full px-3 py-2
border-2 border-gray-300 mb-4 text-sm focus:outline-none focus:border-
gray-900 bg-gray-50"

    onKeyDown={(e) => e.key === 'Enter' && handleSubmit()} />
    {error && <div className="text-red-600 text-xs mb-
3">{error}</div>}}

    <button onClick={handleSubmit}

    className="w-full py-2.5 text-sm font-bold bg-gray-900
text-white border-2 border-gray-900 hover:bg-gray-800 uppercase mb-2">

    {isRegister ? 'Register' : 'Login'}

    </button>

    <button onClick={() => { setIsRegister(!isRegister);
setError(''); }}

    className="w-full text-xs text-gray-500 hover:text-gray-
900 underline">

    {isRegister ? 'Have account? Login' : 'No account?
Register'}}

    </button>

    </div>

  </div>

);
}

```

Файл CardNode.tsx

```

import { memo } from 'react';
import { NodeProps } from 'reactflow';
export const CardNode = memo(({ data }: NodeProps) => {
  return (
    <div className="bg-white border rounded-lg px-4 py-3 cursor-
grab hover:shadow-md transition-shadow"

    style={{ borderColor: data.color, width: 320 }}>

    {data.prompt && <div className="text-[10px] text-gray-400 mb-
1.5 font-medium uppercase tracking-wide">Question</div>}

```

```

        {data.prompt && <div className="text-[11px] text-gray-600 mb-
2 leading-relaxed whitespace-pre-wrap break-words">{data.prompt}</div>}
        {data.prompt && <div className="border-t border-gray-100 mb-
1.5" />}

        <div className="text-[11px] text-gray-800 leading-relaxed
whitespace-pre-wrap break-words">{data.content || '...'}</div>
        </div>
    );
});

```

Файл ContextMenu.tsx

```

import { useState, useEffect, useRef } from 'react';

interface Props {
  x: number; y: number;
  nodeId: string; nodeType: string;
  nodes: any[];
  onClose: () => void;
  onSendInGroup: (groupId: string) => void;
  onDeleteCard: (cardId: string) => void;
  onDeleteGroup: (groupId: string) => void;
  onMoveCard: (cardId: string, groupId: string) => void;
}

export function ContextMenu({ x, y, nodeId, nodeType, nodes,
onClose, onSendInGroup, onDeleteCard, onDeleteGroup, onMoveCard }:
Props) {
  const [promptInput, setPromptInput] = useState('');
  const [showPrompt, setShowPrompt] = useState(false);
  const ref = useRef<HTMLDivElement>(null);

  useEffect(() => {
    const handler = (e: MouseEvent) => {
      if (ref.current && !ref.current.contains(e.target as Node))
onClose();
    };
    document.addEventListener('mousedown', handler);
  });

```

```

const isGroup = nodeType === 'topicGroup';
const isCard = nodeType === 'cardNode';
const groups = nodes.filter(n => n.type === 'topicGroup');

return (
  <div ref={ref} className="absolute z-50 bg-white border
border-gray-300 rounded-lg shadow-xl py-1 min-w-[180px] text-xs font-
mono"

    style={{ left: x, top: y }}>

    {isGroup && (
      <>
        <div className="px-3 py-2 text-[10px] text-gray-400
uppercase">Group: {nodeId.replace('g-', '')}</div>
        <div className="border-t border-gray-100" />

        {!showPrompt ? (
          <button onClick={() => setShowPrompt(true)}
            className="w-full text-left px-3 py-2 hover:bg-
gray-100 text-gray-700">

            </button>
          ) : (
            <div className="px-3 py-2">
              <input autoFocus value={promptInput} onChange={e
=> setPromptInput(e.target.value)}
                onKeyDown={e => { if (e.key === 'Enter') {
onSendInGroup(nodeId); setShowPrompt(false); } }}
                placeholder="Question..." className="w-full px-2
py-1 border border-gray-300 text-xs focus:outline-none focus:border-
gray-500" />
              </div>
            )}
          <button onClick={() => onDeleteGroup(nodeId)}
            className="w-full text-left px-3 py-2 hover:bg-red-
50 text-red-600">

```

```

#Delete group
    </button>
  </>
})

{isCard && (
  <>
    <div className="px-3 py-2 text-[10px] text-gray-400
uppercase">Card</div>
    <div className="border-t border-gray-100" />

    <button onClick={() => onDeleteCard(nodeId)}
      className="w-full text-left px-3 py-2 hover:bg-
red-50 text-red-600">
    </button>

    {groups.length > 0 && (
      <div className="border-t border-gray-100">
        <div className="px-3 py-1 text-[10px] text-
gray-400">Move to group:</div>
        {groups.filter(g => g.id !== nodeId).map(g => (
          <button      key={g.id}      onClick={()      =>
onMoveCard(nodeId, g.id)}
            className="w-full  text-left  px-3  py-2
hover:bg-gray-100 text-gray-700">
            → {(g.data as any).label}
          </button>
        ))}
      </div>
    )}
  </>
)}
</div>
);
}

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.36.001 ПЗ

Лист

80

Файл GroupNode.tsx

```
import React, { memo } from 'react';
import { Handle, Position, NodeProps } from 'reactflow';
const GroupNode = memo(({ data, selected }: NodeProps) => {
  return (
    <div
      className="px-4 py-2 border-2 font-mono text-xs font-
bold uppercase tracking-wider min-w-[150px]"
      style={{
        backgroundColor: data.bgColor || '#f8fafc',
        borderColor: data.color || '#94a3b8',
        color: data.color || '#475569',
      }}
    >
      <Handle type="source" position={Position.Bottom}
className="!bg-gray-400 !w-2 !h-2" />
      <div className="flex items-center gap-2">
        <span>◆</span>
        <span>{data.label}</span>
      </div>
      <div className="text-[10px] font-normal mt-1 normal-
case">
        {data.cardCount} cards
      </div>
    </div>
  );
});
```

Файл ThreadEdge.tsx

```
import { BaseEdge, getSmoothStepPath, Position } from
'reactflow';
export function ThreadEdge({ sourceX, sourceY, targetX,
targetY, style, markerEnd }: any) {
  const [p] = getSmoothStepPath({ sourceX, sourceY, targetX,
targetY,
```

```

        sourcePosition:      Position.Bottom,      targetPosition:
Position.Top, borderRadius: 12 });
        return <BaseEdge path={p} markerEnd={markerEnd} style={{
...style, strokeDasharray: '3,3', strokeWidth: 1 }} />;
    }

```

Файл TopicGroup.tsx

```

import { memo } from 'react';
import { Handle, Position, NodeProps } from 'reactflow';

export const TopicGroup = memo(({ data }: NodeProps) => (
    <div
        className="rounded-lg border-2 w-full h-full flex flex-
col"
        style={{      backgroundColor:      data.bg,      borderColor:
data.color }}
    >
        <Handle      type="target"      position={Position.Top}
className="!bg-gray-300 !w-2 !h-2" />
        <div className="px-3 py-2 border-b" style={{ borderColor:
data.color + '40', backgroundColor: data.bg }}>
            <div      className="text-xs      font-semibold      uppercase
tracking-wider" style={{ color: data.text }}>
                {data.label}
            </div>
            <div className="text-[10px] mt-0.5 opacity-60" style={{
color: data.text }}>
                {data.count} cards
            </div>
        </div>
    </div>
));

```

Приложение Г
(обязательное)

Нейросетевой ассистент с интерактивной панелью
Презентация

					<i>ПГУ 1.090301.36.001 ПЗ</i>	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

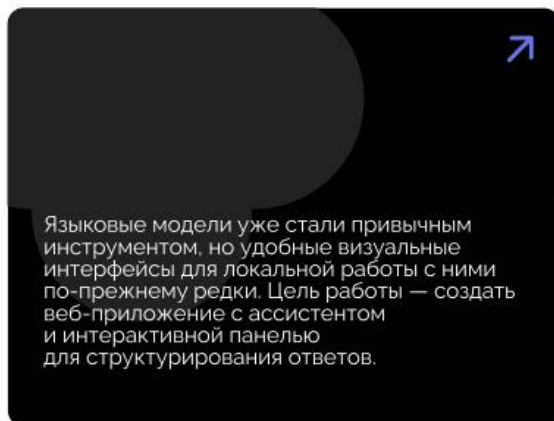


Нейросетевой ассистент с интерактивной панелью

Выпускная квалификационная работа

Автор работы: Дулатов Денис Александрович
Научный руководитель: Леонова Татьяна Юрьевна

Зачем нужен новый формат работы с LLM



2

					ПГУ 1.090301.36.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		84

Задачи работы

↗

01

Серверная часть
Разработать бэкенд часть приложения с интеграцией открытой языковой модели, чтобы обрабатывать запросы пользователя, управлять диалогами и обеспечивать взаимодействие между клиентской частью и хранилищем данных.

02

Клиентский интерфейс
Создать веб-клиент с интерактивной панелью, на которой ответы ассистента можно визуально группировать, перемещать и связывать между собой в удобной для анализа форме.

03

Потоковая генерация
Организовать передачу ответа по мере его формирования через WebSocket, чтобы пользователь видел результат без ожидания завершения полной генерации текста моделью.

04

Категоризация ответов
Реализовать автоматическое определение тематики ответа и его размещение в соответствующей группе на панели, обеспечив логичную визуальную структуру диалога.

05

Хранение контекста
Сохранить историю взаимодействий в базе данных и использовать накопленные факты при последующих запросах, чтобы поддерживать связность беседы при ограничении длины контекста.

Архитектура приложения

↗

```
graph LR
    subgraph Client [Клиентская часть]
        Browser[Браузер]
    end
    subgraph Server [Сервер и данные]
        FastAPI[FastAPI]
        SQLite[(SQLite)]
    end
    subgraph Integration [Нейросетевая интеграция]
        LLM[LLM Saiga]
    end
    Browser -- "REST API" --> FastAPI
    Browser -- "WebSocket" --> FastAPI
    FastAPI <--> SQLite
    FastAPI -- "API" --> LLM
```

- Клиентская часть**
Браузер предоставляет доступ к интерфейсу пользователя, формирует запросы к серверу и отображает потоковые ответы, карточки диалогов и связи между ними.
- Сервер и данные**
FastAPI обрабатывает REST-запросы и WebSocket-соединения, а SQLite используется для хранения диалогов, сообщений и служебных данных приложения.
- Нейросетевая интеграция**
LLM Saiga подключается локально и участвует в генерации ответов без обращения к облачным API, что упрощает контроль над данными и средой запуска.
- Развёртывание и сборка**
Docker Compose обеспечивает воспроизводимый запуск компонентов, а GitHub Actions автоматизирует сборку и проверку проекта при внесении изменений.

Технологический стек



Слой	Технологии
Бэкенд	Python 3.13, FastAPI
Фронтенд	TypeScript, React, Vite, Tailwind CSS
Панель	React Flow
База данных	SQLite, SQLAlchemy
Нейросеть	Salga Gemma 3, llama.cpp
Развёртывание	GitHub Actions

На основе реализованного состава проекта

Основные технологии проекта и их роль в реализации локального веб-приложения.

Стек охватывает весь цикл: интерфейс, сервер, хранение данных, локальный запуск модели и автоматизацию развёртывания.

Сценарии использования системы

Логика работы пользователя, интерфейса и нейросетевого ассистента в рамках основных функций приложения.

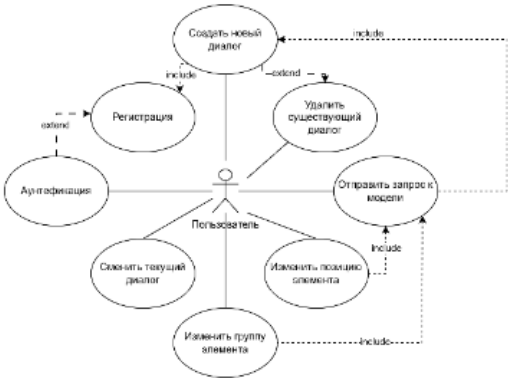
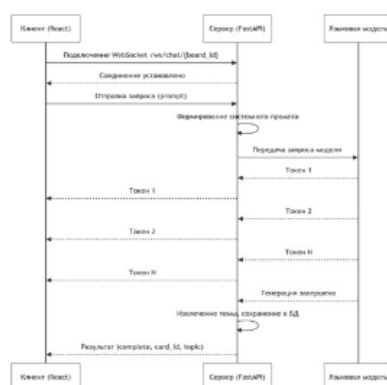


Схема передачи по WebSocket соединению



7

Алгоритм категоризации ответов



8

Основные элементы интерфейса

История

Список диалогов

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

Ввод

Поле ввода

История

Список диалогов

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

История

Список диалогов

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

История

Список диалогов

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

Список диалогов

Боковая панель позволяет быстро переключаться между диалогами, создавать новые и удалять ненужные, сохраняя удобную навигацию по истории общения.

Поле ввода

В нижней части окна расположен ввод запроса, откуда пользователь отправляет сообщение ассистенту и при необходимости останавливает генерацию ответа.

Интерактивная панель

Центральная область отображает карточки ответов в виде свободно размещаемых блоков, что упрощает обзор структуры беседы и выделение ключевых фрагментов.

Связи между блоками

Карточки можно перемещать и соединять линиями, формируя наглядные тематические группы и связи между отдельными ответами внутри одного диалога.

Итоги работы



Создано клиент-серверное веб-приложение с локальной LLM, потоковой генерацией, фактологическим сжатием контекста и автоматической категоризацией ответов.

Спасибо за внимание

