МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-приложения для запросов на генерацию текста с применением API YandexGPT и технологии блокчейн

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-514.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ А.Т. Латипова

Автор работы,
студент группы КЭ-402
_____ Н.П. Никитин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

студенту группы КЭ-402
Никитину Никите Павловичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. Тема работы (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка веб-приложения для запросов на генерацию текста
с применением API YandexGPT и технологии блокчейн.
2. Срок сдачи студентом законченной работы: 03.06.2024 г.
3. Исходные данные к работе
3.1. Прасти Н. Блокчейн. Разработка приложений. // БХВ-Петербург, 2018. – 256 с.
3.2. Metamask. [Электронный ресурс] URL: https://metamask.io/ (дата обращения: 05.02.2024 г.).
3.3. YandexGPT. [Электронный ресурс] URL: https://cloud.yandex.ru/ (дата обращения: 07.02.2024 г.).
4. Перечень подлежащих разработке вопросов
4.1. Провести анализ предметной области.
4.2. Выполнить обзор существующих аналогов.
4.3. Спроектировать веб-приложение.
4.4. Реализовать спроектированное веб-приложение.
4.5. Протестировать реализованное веб-приложение.
5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. А.Т. Латипова

Задание принял к исполнению Н.П. Никитин

ГЛОССАРИЙ

1. *Блокчейн* – это особая структура данных, применяемая для создания децентрализованного регистра, которая состоит из блоков (block), особым образом соединенных в цепочку (chain). Каждый блок содержит набор транзакций, хеш предыдущего блока, метку времени (время создания блока), сумму отчисления майнеру за созданный блок и т.д. Поскольку каждый блок содержит хеш предыдущего блока, они связаны в цепочку. Каждый узел сети хранит полную или облегченную копию блокчейн [1].

2. *Смарт-контракт Ethereum* – это программа, которая выполняется в сети Ethereum и работает исключительно так, как запрограммировано, без риска простоя, цензуры, мошенничества и вмешательства третьей сторон [1].

3. *Ethereum* – это децентрализованная платформа, на которой можно запускать приложения в виде смарт-контрактов. Приложение может состоять из одного или нескольких смарт-контрактов [1].

4. EthereumWallet – это приложение клиент-пользовательского интерфейса Ethereum, позволяющее создавать аккаунт, отправлять эфир, разворачивать контракты, вызывать методы контрактов и многое другое [1].

5. *NFT (NonFungibleToken)* – невзаимозаменяемый NFT токен [2].

6. *NFT токен* – единица учета в блокчейн [2].

7. *Metamask* – криптовалютный кошелек и шлюз для блокчейн приложений [3].

8. *Чат-бот AI (искусственный интеллект)* – это программа, которая использует искусственный интеллект для взаимодействия с пользователями через чат. Он способен анализировать и понимать вопросы пользователей, предоставлять информацию или решать задачи. Нейронные модели с искусственным интеллектом могут быть использованы в различных областях, таких как обслуживание клиентов, продажи, образование и многое другое [3].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

На сегодняшний день актуальность чат-ботов с искусственным интеллектом активно растет, и является активно развивающимся направлением [2]. Они представляют собой программное обеспечение, способное вести диалог с людьми или другими ботами через текстовые или устные сообщения. Нейронные модели могут быть использованы для различных целей, включая обслуживание клиентов, автоматизацию задач, обучение и развлечения.

Благодаря использованию искусственного интеллекта, нейронные модели могут обучаться на основе предыдущих взаимодействий с пользователями и становиться все более умными и адаптивными. Служат для помощи в различных областях, начиная от обслуживания клиентов и поддержки до проведения опросов и продаж [1].

Преимущества чат-ботов включают повышение эффективности обслуживания клиентов, сокращение времени решения проблем, увеличение доступности услуг, а также сокращение операционных издержек для компаний. Кроме того, нейронные модели могут быть настроены, для выполнения широкого спектра задач, начиная от простых запросов до сложных операций.

С развитием технологий искусственного интеллекта, нейронные модели, становятся все более умными и способными адаптироваться к потребностям пользователей. Они играют важную роль в улучшении опыта пользователей, оптимизации бизнес-процессов и создании новых возможностей для взаимодействия с технологиями.

Использование чат-ботов подразделяются на различные области, такие как обслуживание клиентов, автоматизация задач, обучение и развлечение. Эти приложения продолжают развиваться и находить новые области применения благодаря своей гибкости и способности адаптироваться к потребностям пользователей.

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-приложения, с применением модели искусственного интеллекта Yandex GPT и технологии блокчейн. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор существующих решений;
2) определить требования веб-приложения;
3) спроектировать веб-приложение;
4) реализовать веб-приложение;
5) провести тестирование веб-приложения.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 90 страниц, объем списка литературы – 27 источников.

В первом разделе описывается постановка задачи, а также предметная область. Также произведен анализ предметной области и выбор инструментов и технологий для реализации.

Вторая глава посвящена анализу требований к разрабатываемой системе, также в ней содержится диаграмма вариантов использования и база данных веб-приложения.

В третьей главе приведена архитектура разрабатываемой системе, в ней содержится 3 диаграммы – диаграмма последовательности, диаграмма деятельности и диаграмма компонентов.

Четвертая и пятая главы посвящены реализации и тестированию системы.

В приложении А содержится спецификация диаграммы вариантов использования.

В приложении Б содержатся скриншоты веб-приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

Целью данной работы является разработка веб-приложения для запросов на генерацию текста с помощью нейронной модели и технологии блокчейн. Веб-приложение – клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения являются межплатформенными службами.

Анализ аналогичных проектов

Для анализа готовых проектов были выбраны два веб-приложения:

1) ChatGPT;
2) GigaChat.

Веб-приложения содержат в себе идентичный функционал – интерфейс каждого из представленных приложений содержит в себе возможность подключить аккаунт и создавать запрос с текстом. В процессе изучения веб-приложений, был проведен анализ ранее упомянутых чат-ботов с искусственным интеллектом.

ChatGPT

ChatGPT – это модель генерации текста, созданная компанией OpenAI. Она способна генерировать естественно звучащие ответы на вопросы и участвовать в разговорах на естественном языке. Модель основана на искусственном интеллекте и обучена на больших объемах текстовых данных, чтобы лучше понимать и генерировать человекоподобные ответы. Это мощный инструмент для коммуникации и общения с компьютером, а также для выполнения различных задач, требующих обработки и генерации текста.

Официально ChatGPT недоступен для использования в России, необходимо будет идти обходным способом, чтобы пройти авторизацию. Есть несколько вариантов регистрации: Создать полностью новый аккаунт или пройти авторизацию через Google, Microsoft, Apple ID. На рисунке 1 изображен интерфейс авторизации приложения.

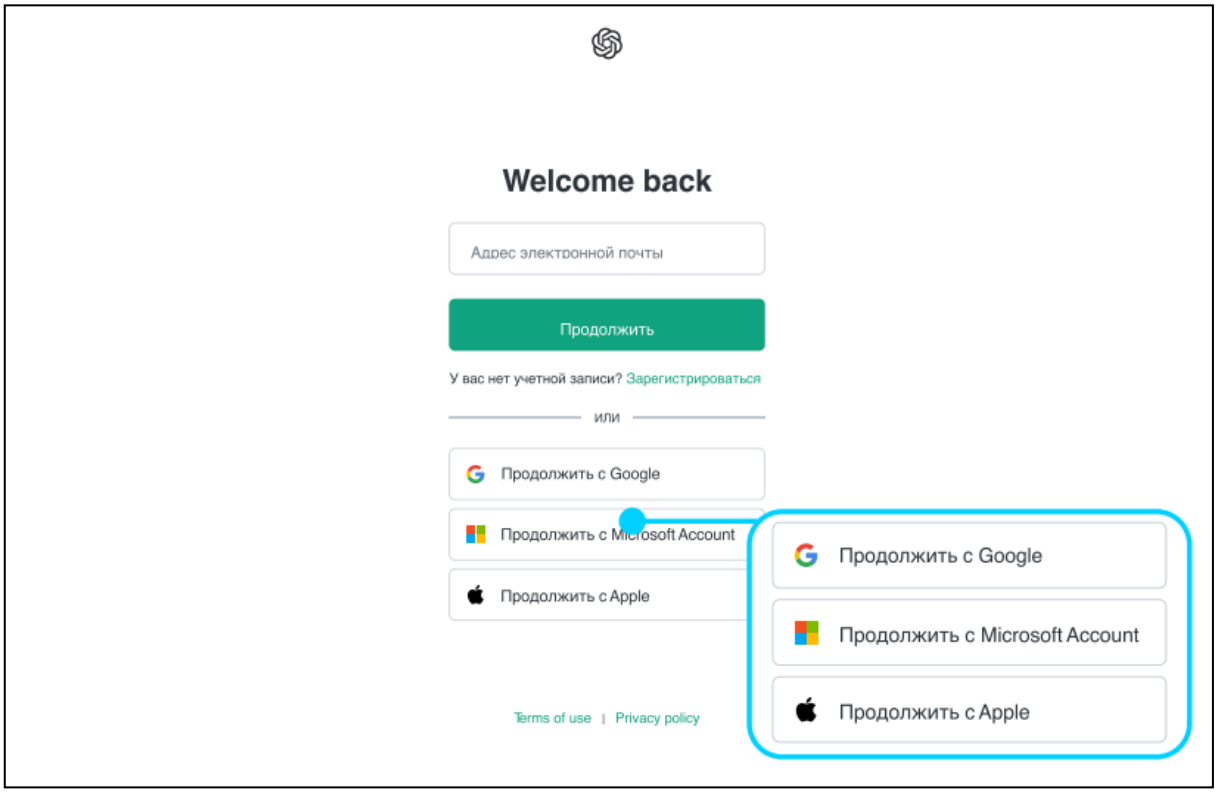

Рисунок 1 – Интерфейс авторизации веб-приложения

Обзор интерфейса

Интерфейс очень простой – все расположено в одном окне. После авторизации вы сразу попадаете в диалоговое окно, в котором есть начальный вопрос. ChatGPT сразу предлагает готовые варианты того, что можно спросить. Для того, чтобы написать запрос, вам нужно нажать на область чата и вписать то, что вы хотите узнать, далее нажать «Enter» и модель начнет отвечать. На рисунке 2 изображено диалоговое окно веб-приложения.

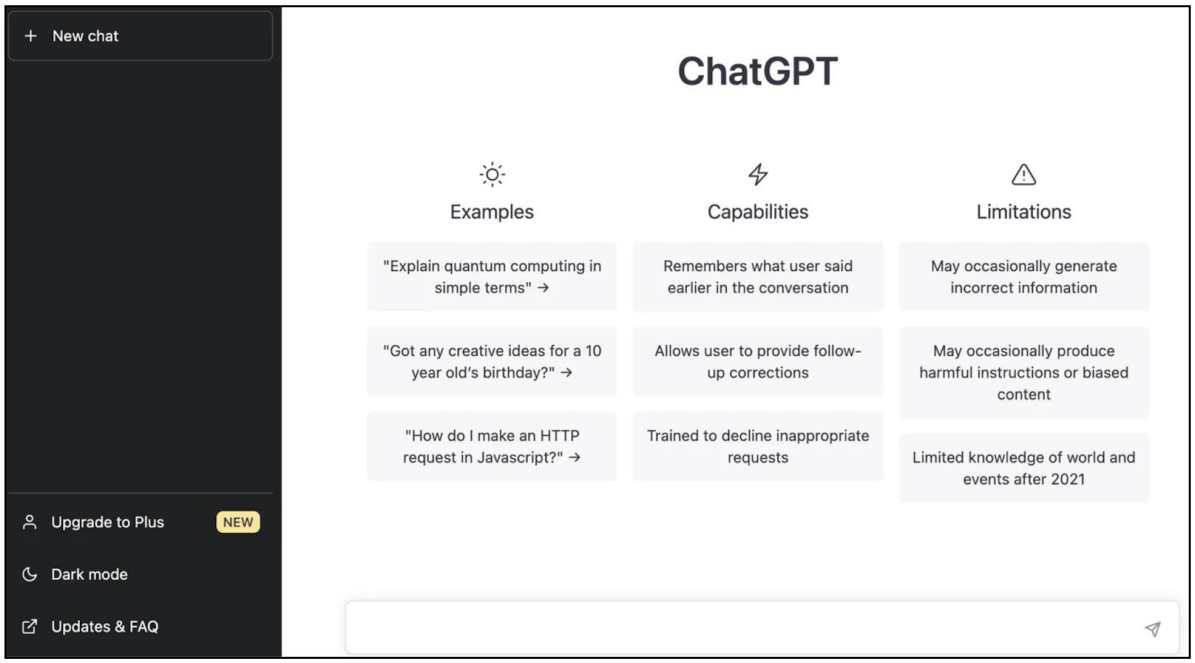

Рисунок 2 – Диалоговое окно веб-приложения

Под ответом есть четыре иконки: скопировать в буфер обмена весь ответ, лайк, открывающий окно для позитивной обратной связи по ответу, дизлайк, где можно подробнее объяснить, почему ответ был некорректен, либо выбрать готовые варианты. На рисунках 3 и 4 изображено окно позитивной и негативной оценки обратной связи по ответу.

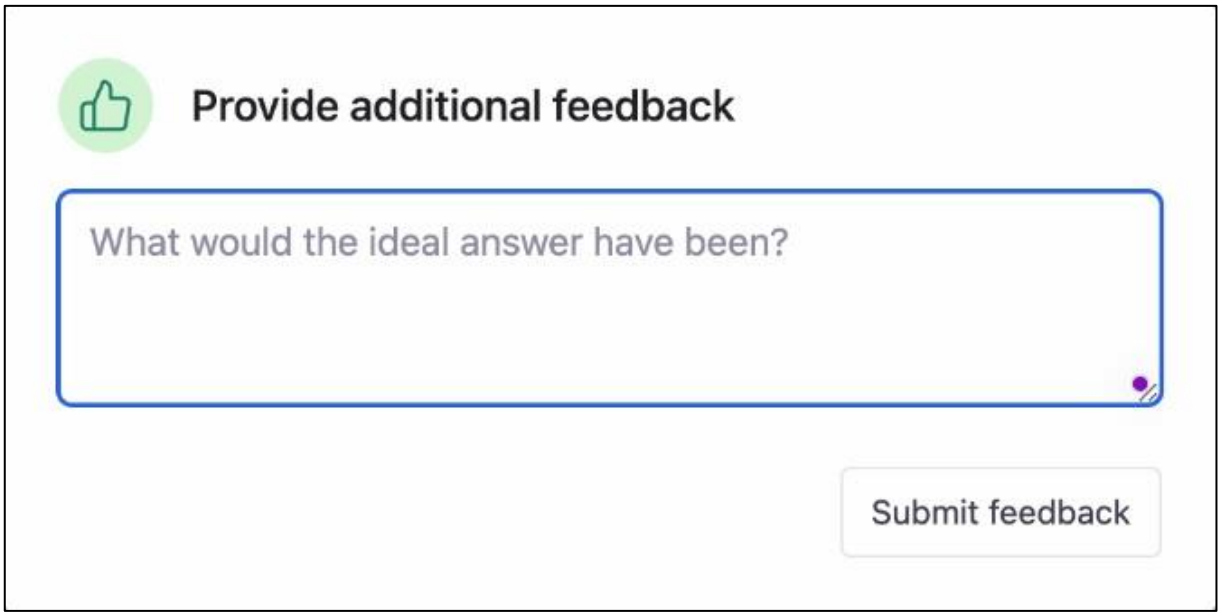

Рисунок 3 – Интерфейс окна позитивной оценки

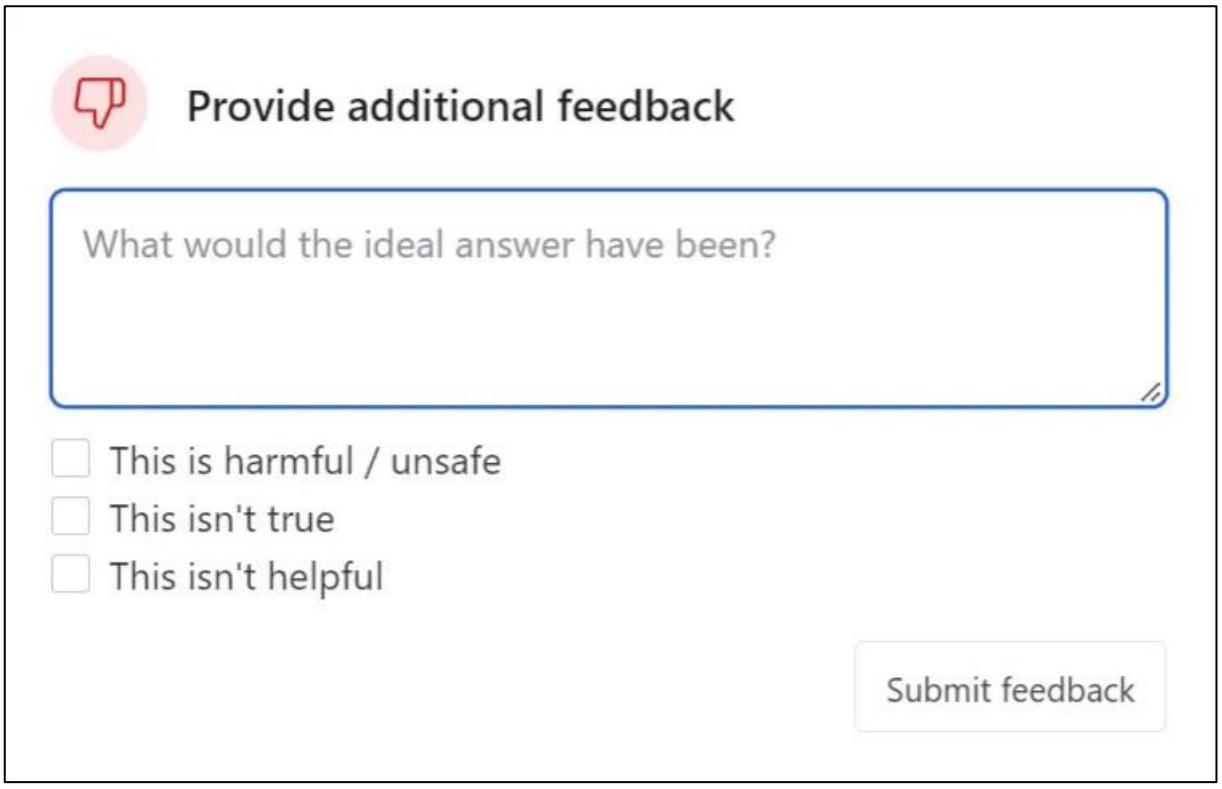

Рисунок 4 – Интерфейс окна негативной оценки

После добавления нового ответа, чат попросит дать оценку новой генерации. На рисунке 5 изображено окно оценки новой генерации.

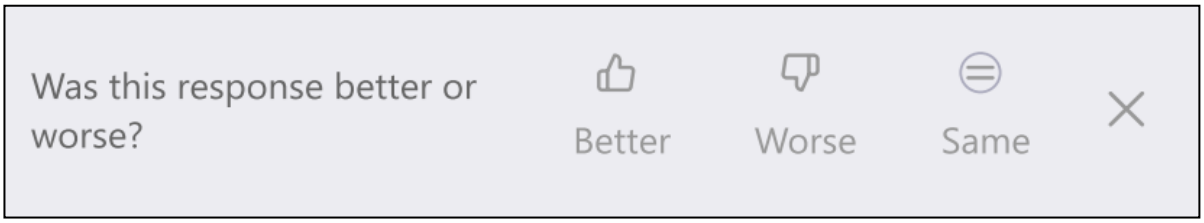

Рисунок 5 – Интерфейс окна оценки новой генерации

Следующий элемент интерфейса – меню, в нем отображаются все рабочие чаты, статус подписки с опцией купить платную версию и общие настройки.

Чтобы создать новый чат нужно нажать «new chat». Как только его создадите, он появится в списке в боковом меню. Каждая сессия в чате будет записываться, и вы сможете вернуться к ним спустя время и продолжить общение.

Следующая кнопка – «upgrade plan». При нажатии на нее, вы увидите текущий тариф, также можно купить платную подписку. На рисунке 6 изображен интерфейс выбора тарифа подписки.

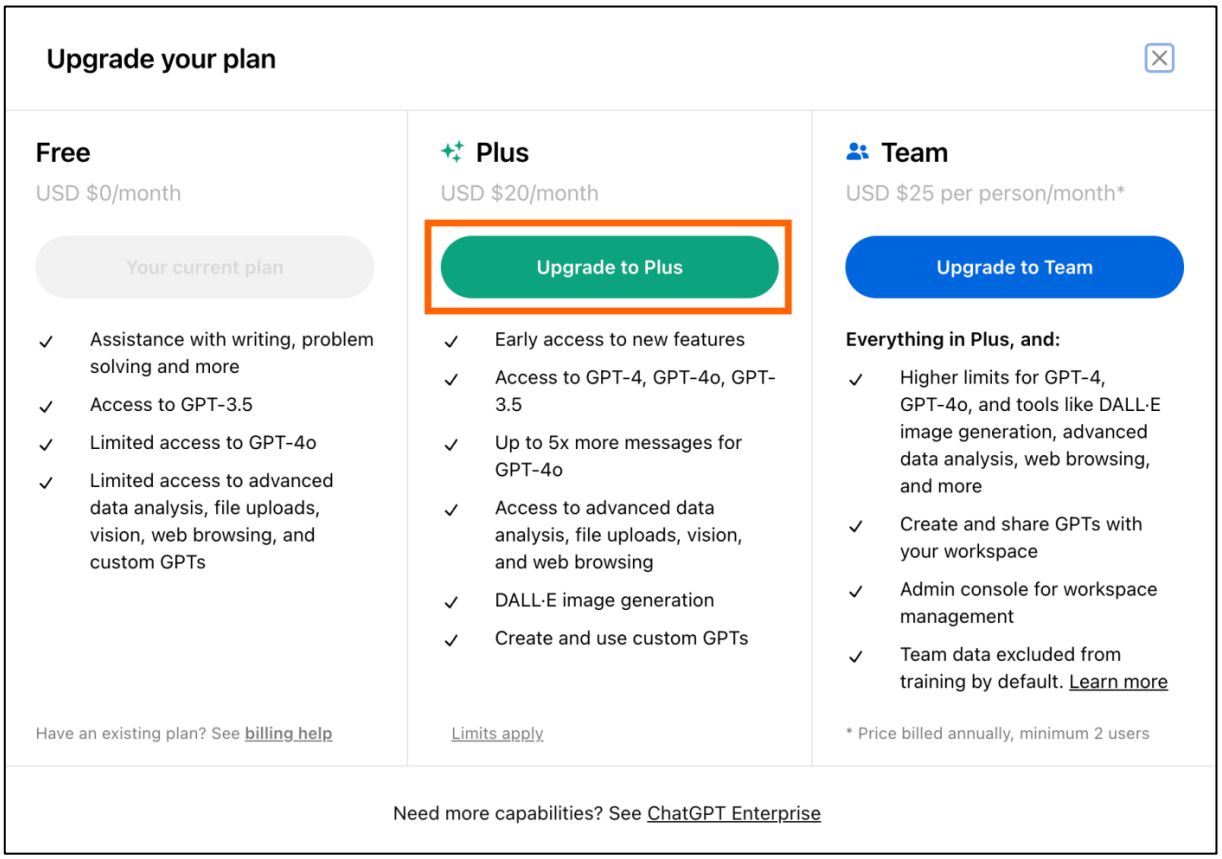

Рисунок 6 – Интерфейс выбора тарифа подписки

При нажатии на учетную запись, появится меню со следующими настройками: пользовательские настройки, общие настройки, выход из учетной записи. На рисунке 7 изображен интерфейс выбора настроек.

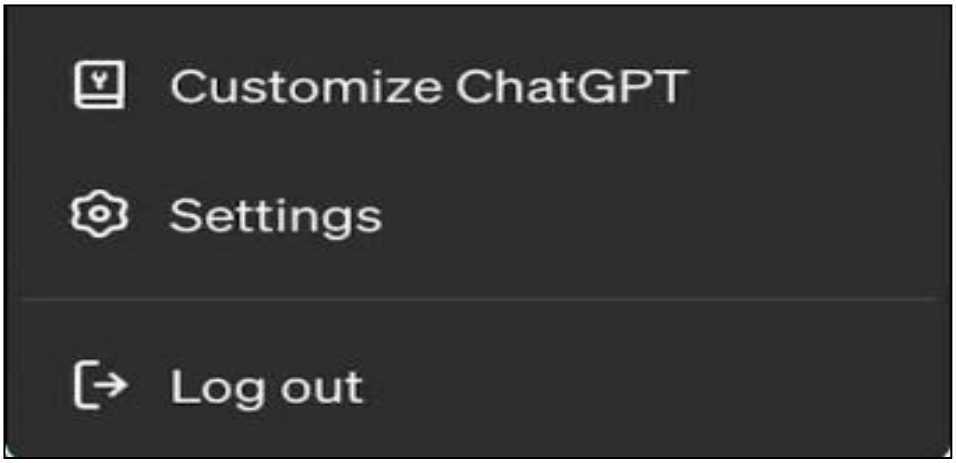

Рисунок 7 – Интерфейс выбора тарифа настроек

В пользовательских настройках можно указать важную информацию, которой бы хотели поделиться с чатом, а также в каком виде хотите получать ответы. На рисунке 8 изображен интерфейс пользовательских настроек веб-приложения.

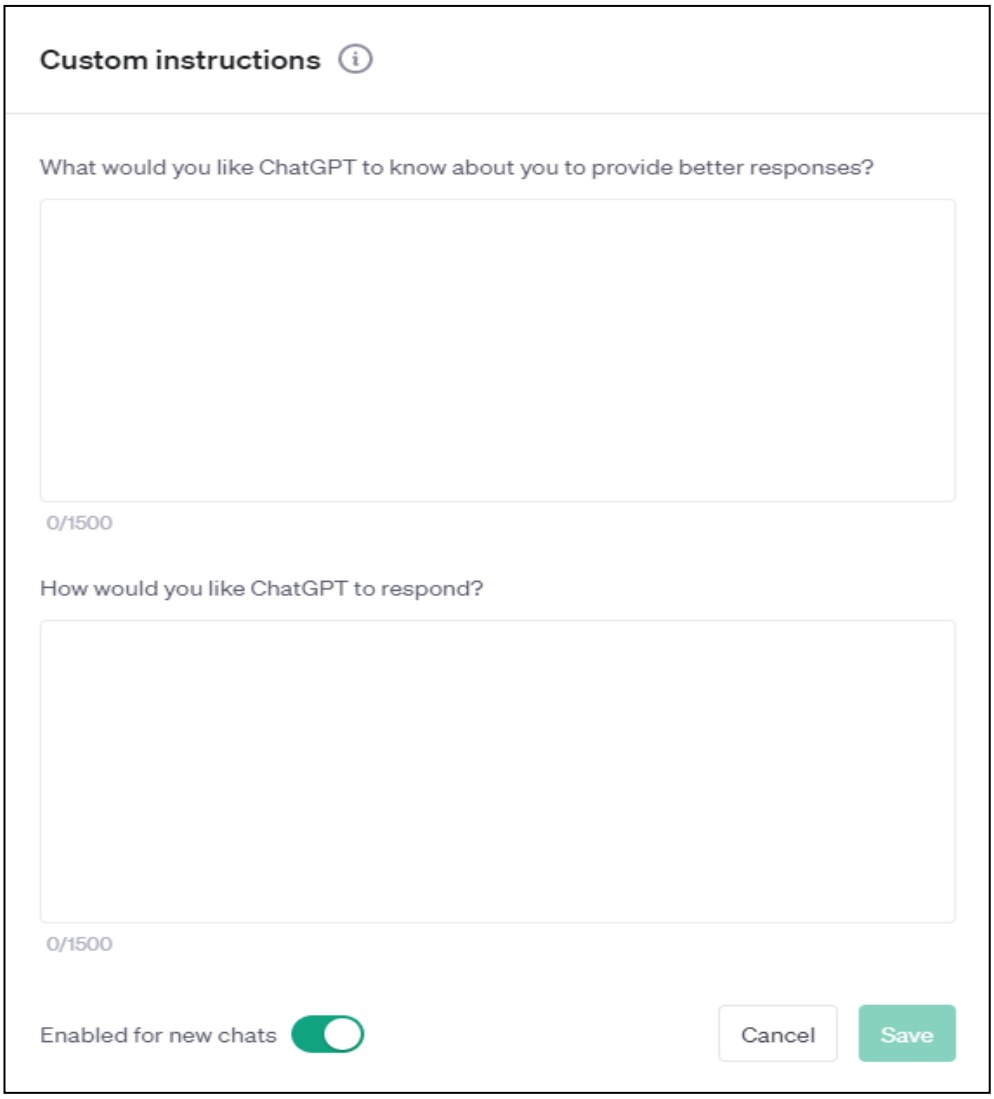

Рисунок 8 – Интерфейс пользовательских настроек

В общих настройках можно выбрать цвет темы, локализацию, просмотреть и удалить архивные чаты. Также в настройках есть возможность очищения данных. На рисунке 9 изображен интерфейс общих настроек веб-приложения.

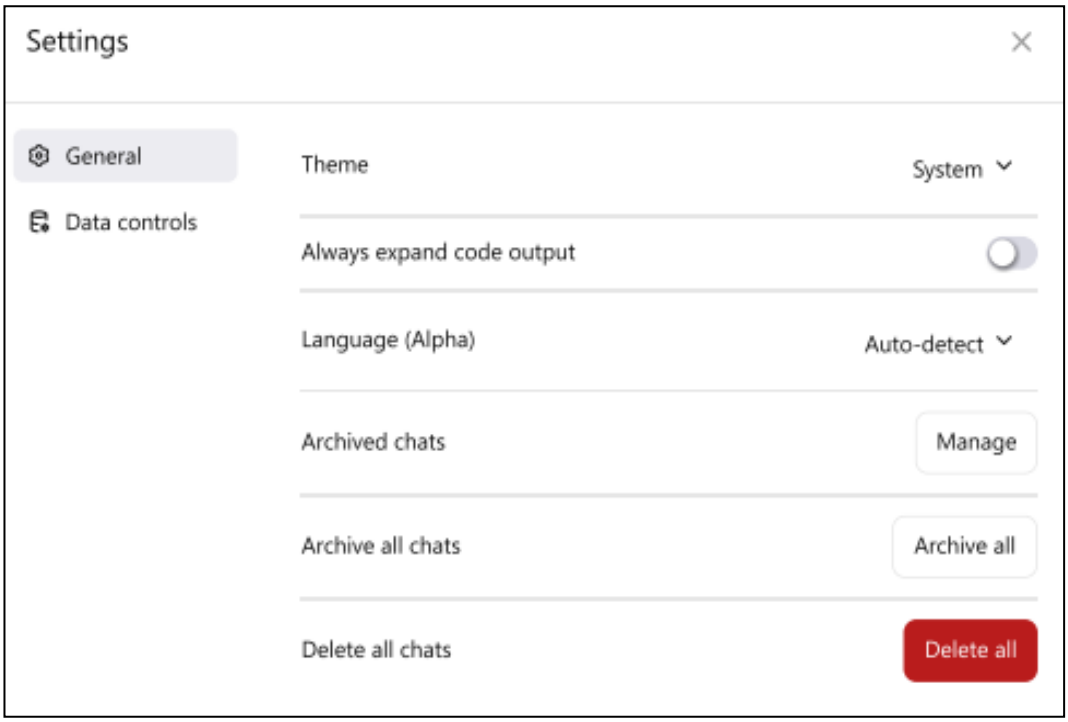

Рисунок 9 – Интерфейс общих настроек

Обзор интерфейса GigaChat

После авторизации пользователь попадает на главную страницу, на которой находится весь функционал (рисунок 10).

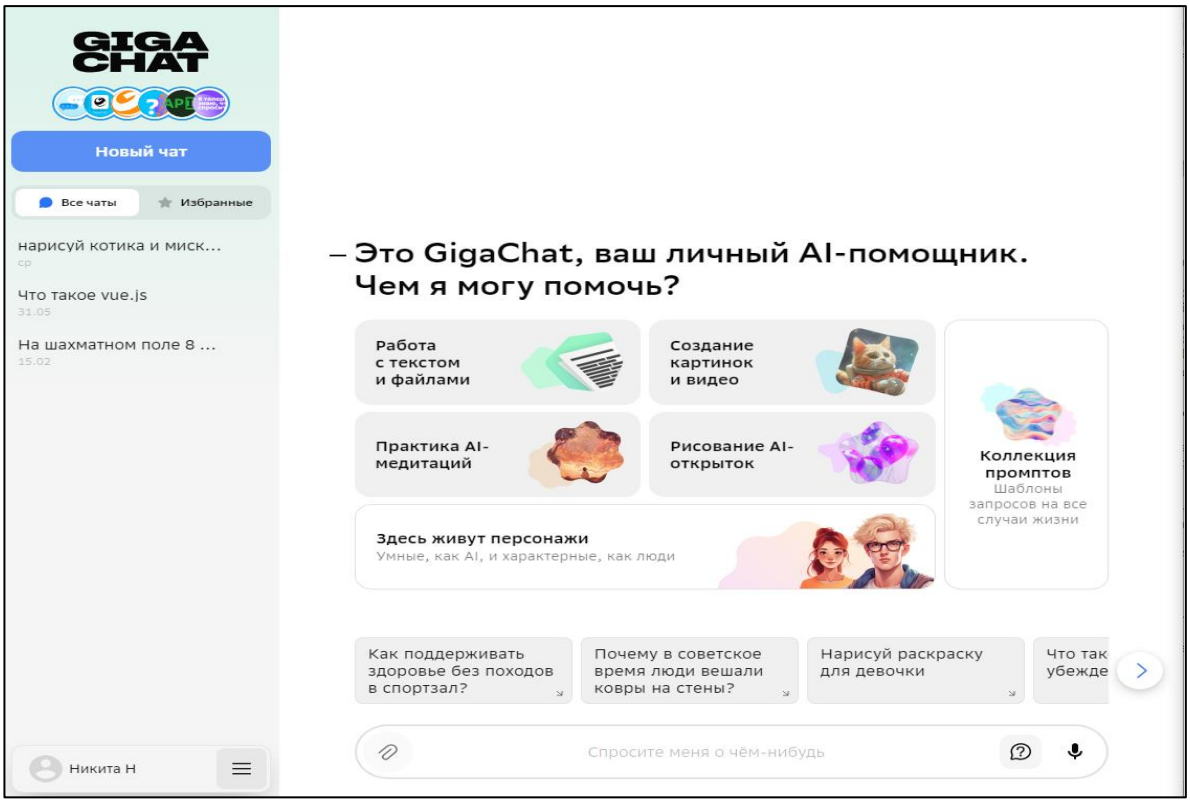

Рисунок 10 – Интерфейс главной страницы

На первом экране видим приветствие от GigaChat, который предлагает сразу перейти к делу. Для этого достаточно написать любой вопрос или просьбу. Также есть возможность создавать новые чаты и добавлять их в избранное. GigaChat может генерировать изображения и тексты в одном окне. На рисунке 11 представлен интерфейс генерации изображения при помощи нейросети kandinsky 2.2.

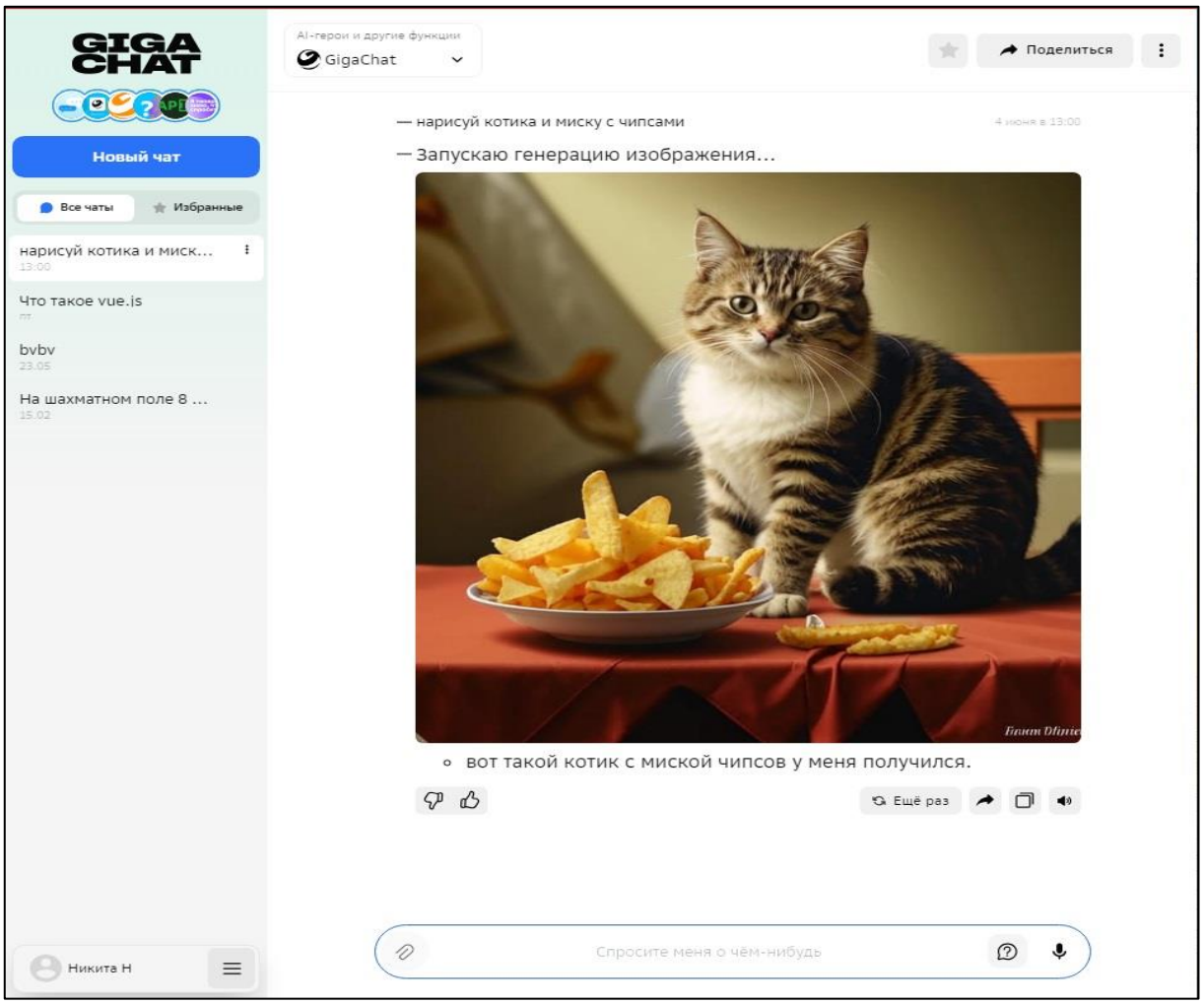

Рисунок 11 – Интерфейс генерации изображения

Не так давно, в Gigachat, появилась функция загрузки текстовых файлов, которая изображена на рисунке 12. Для того, чтобы воспользоваться функцией, необходимо перетащить или выбрать файл. После этого, нажать на кнопку прикрепления в новый чат, и нейросеть начнет обрабатывать запрос прикрепленного файла.

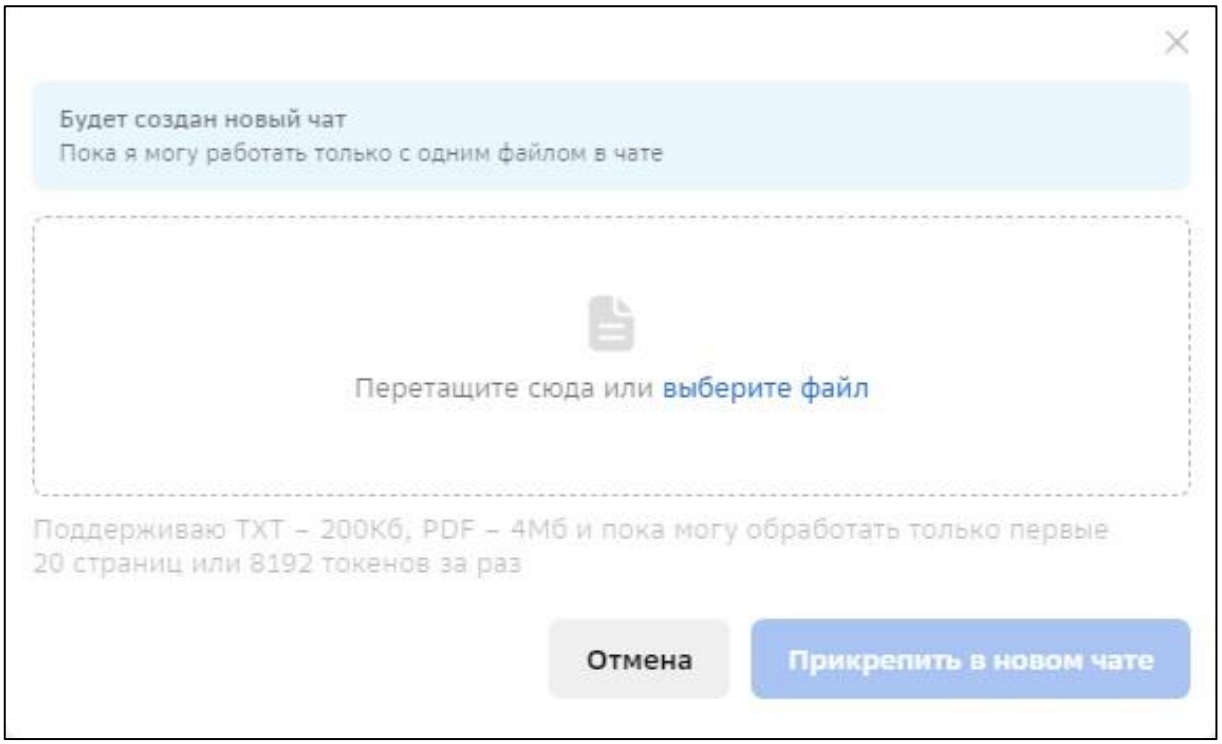

Рисунок 12 – Интерфейс функции загрузки текстовых файлов

Имеется возможность голосового ввода – для этого надо нажать на значок микрофона в окошке сообщения.

Каждому ответу можно ставить лайк или дизлайк. Эти данные передаются разработке и на основе пользовательского опыта генерации улучшаются.

Справа от генерации ответа расположены три иконки: круглые стрелочки означают новую генерацию ответа, стрелка – с ее помощью вы можете делиться результатами в формате «запрос – ответ» в Telegram, по прямой ссылке или с помощью QR-кода, нажав на третью кнопку, вы скопируете диалог в буфер обмена.

Слева находится основное меню. Рядом с логотипом GigaChat расположены значки – это истории, в которых рассказываются основные принципы работы с сервисом.

Ниже логотипа большая синяя кнопка – создания нового чата. Каждый раз, когда необходимо пообщаться на новую тему, создавайте отдельный чат, чтобы темы не смешивались.

Список всех чатов находится по порядку их использования. Новый чат всегда поднимается вверх.

Также рядом с каждым чатом есть 3 точки. Можно поделиться всем чатом напрямую, добавить его в избранное или удалить. Эти же функции продублированы в самом окне чата справа вверху.

В самом низу меню находятся «FAQ» – база знаний. Интерфейс базы знаний представлен на рисунке 13.

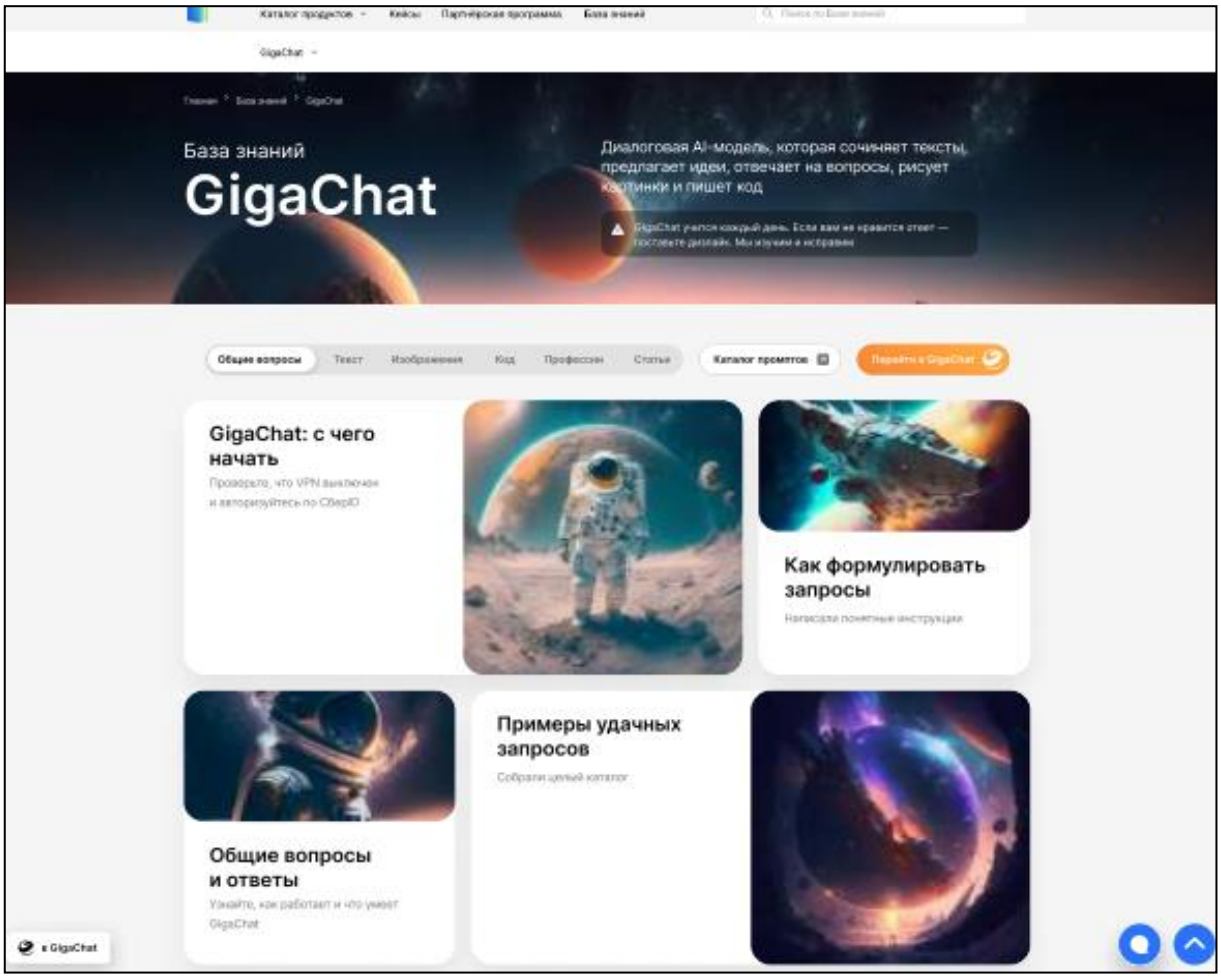

Рисунок 13 – Интерфейс базы знаний веб-приложения

«Поддержка» – можно описать свою проблему и отправить ее в телеграм-бот.

«О GigaChat» – по ссылке находится статья о том, что такое GigaChat. Интерфейс представлен на рисунке 14.

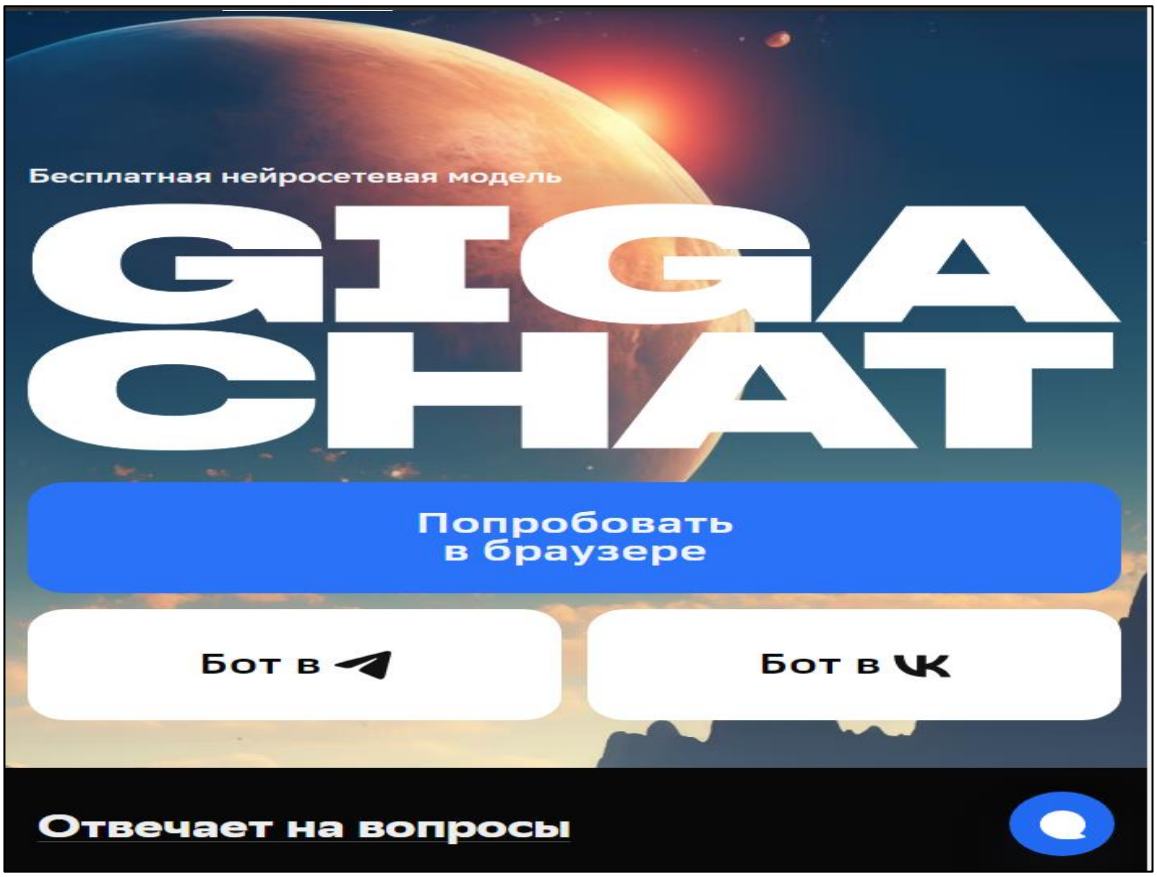

Рисунок 14 – Интерфейс статьи Gigachat

Проанализировав приложения, можно выделить общие черты:

1) в веб-приложениях в основном имеется клиент-серверной части;

2) веб-приложения работают после авторизации аккаунта, личного кабинета пользователя;

3) веб-приложения работают с технологией распознавания голоса.

1.2. Обзор средств разработки

Фронтенд и бэкенд это две основные части веб-приложений, каждая из которых выполняет определенные функции и имеет свои отличительные особенности.

Фронтенд отвечает за то, как пользователи взаимодействуют с веб-приложением. Это часть приложения, которую пользователь видит, и с которой он взаимодействует. Фронтенд включает в себя визуальные и интерактивные элементы, такие как веб-страницы, кнопки, формы,

изображения, анимации и так далее. Технологии фронтенда обычно включают HTML, CSS и JavaScript, а также фреймворки и библиотеки, такие как React, Angular, Vue.js и другие. Основная задача фронтенда обеспечить удобство использования и интерактивность веб-приложения для пользователя.

React

Это библиотека JavaScript, которая применяется для создания пользовательских интерфейсов в веб-приложениях. React предоставляет эффективные инструменты для разработки интерактивных и динамических пользовательских интерфейсов. Она позволяет разработчикам создавать компоненты, которые обновляются автоматически при изменении данных.

React имеет несколько ключевых особенностей.

1. React организован вокруг концепции компонентов, которые могут быть переиспользованы и объединены для построения сложных пользовательских интерфейсов.

2. React использует виртуальный DOM для эффективного обновления интерфейса при изменении данных, что улучшает производительность и позволяет избежать лишних обновлений интерфейса.

3. React использует JSX, специальный синтаксис, который позволяет интегрировать HTML-подобный код непосредственно в JavaScript, делая написание компонентов более выразительным и удобным.

4. React поощряет односторонний поток данных, где данные передаются вниз по иерархии компонентов, что обеспечивает простоту и прозрачность управления состоянием.

5. В более поздних версиях React были добавлены функциональные компоненты и хуки, что делает написание компонентов более простым и удобным [12].

Vue.js

Открытый JavaScript фреймворк, который используется для создания пользовательских интерфейсов и одностраничных приложений. Он позволяет создавать динамические интерфейсы с помощью компонентов, реактивных данных и простого синтаксиса. Vue.js широко используется веб-разработчиками для создания современных и отзывчивых интерфейсов.

Vue.js имеет несколько ключевых особенностей.

1. Vue.js обеспечивает реактивное обновление данных – когда данные изменяются, пользовательский интерфейс автоматически обновляется, не требуя явного управления.

2. В Vue.js приложения строятся из множества компонентов, каждый из которых может содержать свою логику, представление и стили, что способствует повторному использованию кода и упрощает поддержку.

3. Синтаксис Vue.js прост в использовании, что позволяет быстро создавать сложные пользовательские интерфейсы. В то же время фреймворк обладает гибкостью, позволяя разработчикам использовать только те функции, которые им нужны.

4. Существует множество официальных плагинов и библиотек, которые помогают расширить функциональность Vue.js и упростить разработку.

Next.js

Это фреймворк React для создания универсальных приложений на основе JavaScript. Он предоставляет разработчикам инструменты для создания статических и динамических веб-приложений с использованием React, а также предоставляет решения для серверного рендеринга, предзагрузки данных и маршрутизации. Next.js имеет встроенную поддержку многих современных технологий, таких как TypeScript, CSS-in-JS и API-роутинг, что делает его популярным выбором для разработки веб-приложений.

Преимущества Next.js представлены ниже.

1. Рендеринг на стороне сервера. Next.js позволяет выполнять SSR (серверный рендеринг), что улучшает производительность и SEO-оптимизацию веб-приложения.

2. Простота использования. Next.js предоставляет простой и интуитивно понятный API для работы с роутингом, предварительной загрузкой и другими функциями.

3. Встроенная поддержка статической генерации. Next.js позволяет генерировать статические файлы приложения, что упрощает его развертывание и улучшает производительность.

4. Расширяемость. Next.js обладает богатой экосистемой плагинов и инструментов, которые позволяют расширять его функциональность по мере необходимости.

Недостатки Next.js представлены ниже.

1. Сложность конфигурации. Некоторые пользователи могут столкнуться с трудностями при настройке среды разработки и конфигурации проекта на Next.js.

2. Ограниченные возможности кастомизации. При использовании некоторых функций Next.js может возникнуть ограничение в кастомизации поведения и внешнего вида приложения.

3. Возможные проблемы с производительностью. Неправильное использование SSR или других функций Next.js может привести к проблемам с производительностью веб-приложения [22].

SCSS

Это надстройка над обычным CSS, предоставляющая дополнительные возможности и функциональность для работы с каскадными таблицами стилей. SCSS позволяет использовать переменные, вложенность, миксины, наследование и другие возможности для создания более чистого и модульного кода стилей. После написания кода на SCSS,

его нужно скомпилировать в обычный CSS перед тем, как использовать его на веб-странице.

Бэкенд отвечает за обработку данных и бизнес-логику веб-приложения. Это часть приложения, которая работает вне обзора пользователя. Бэкенд обрабатывает запросы пользователя, взаимодействует с базами данных, обеспечивает безопасность и управляет всеми аспектами, связанными с функциональностью и обработкой данных. Языки программирования и фреймворки бэкенда обычно включают Node.js, Python, Java, Ruby, PHP, .NET и многие другие [14].

TypeScript

Это высокоуровневый язык программирования, который является надмножеством JavaScript. Он добавляет статическую типизацию, классы, интерфейсы и другие концепции объектно-ориентированного программирования к языку JavaScript. TypeScript компилируется в стандартный JavaScript, что позволяет использовать его на любой платформе, где работает JavaScript.

Преимущества TypeScript представлены ниже.

1. Статическая типизация. TypeScript позволяет устанавливать типы данных для переменных, функций и объектов, что помогает обнаруживать ошибки на этапе компиляции.

2. Улучшенная поддержка инструментов. TypeScript интегрируется хорошо с популярными средами разработки, такими как Visual Studio, VS Code, Sublime и другими, предоставляя мощные инструменты для отладки и рефакторинга кода.

3. Читаемость и поддержка кода. TypeScript позволяет писать более понятный и чистый код благодаря типизации, что упрощает поддержку и дальнейшее развитие проекта.

4. Компиляция в JavaScript. TypeScript компилируется в обычный JavaScript, что значит, можно использовать его в любом проекте, где уже используется JavaScript.

Недостатки TypeScript представлены ниже.

1. Дополнительный шаг компиляции. Использование TypeScript требует дополнительного этапа компиляции, что может замедлить процесс разработки и увеличить время обновления кода.

2. Необходимость изучить новый язык. Для тех, кто уже знаком с JavaScript, могут потребоваться дополнительные усилия для изучения синтаксиса TypeScript.

3. Сложность внедрения на существующие проекты. Переход на TypeScript для уже существующих проектов может быть сложен из-за необходимости переписать уже существующий код [25].

Node.js

Это среда выполнения JavaScript, построенная на движке Chrome V8. Она позволяет выполнять JavaScript на стороне сервера, что открывает возможности для создания масштабируемых и быстрых сетевых приложений. Node.js также обеспечивает доступ к API операционной системы и файловой системе, что делает его полезным инструментом для создания серверных приложений, API и других приложений. Node.js широко используется веб-разработчиками для создания серверной части приложений на JavaScript.

Преимущества Node.js представлены ниже.

1. Node.js работает на основе событийного цикла, что позволяет обрабатывать множество запросов эффективно и масштабировать приложения.

2. Использование JavaScript как языка программирования как на клиентской, так и на серверной стороне упрощает разработку и поддержку приложений.

3. Node.js имеет огромное сообщество разработчиков, что приводит к широкому выбору пакетов и модулей для различных нужд, доступных через npm (Node Package Manager).

4. Встроенная поддержка технологий, таких как WebSocket, позволяет легко создавать веб-приложения, основанные на взаимодействии в режиме реального времени.

Недостатки Node.js представлены ниже.

1. Из-за однопоточной модели выполнения, долгие операции ввода-вывода могут блокировать цикл событий и привести к недоступности сервера для других запросов.

2. Иногда изменения в API могут вызвать несовместимость с более ранними версиями, что может затруднить обновление приложений.

3. В силу особенностей однопоточной модели выполнения, Node.js может быть менее подходящим для задач, требующих интенсивных вычислений.

Java

Это высокоуровневый объектно-ориентированный язык программирования, изначально разработанный компанией Sun Microsystems (позднее приобретенной компанией Oracle). Java широко используется для создания мобильных приложений, веб-приложений, встраиваемых систем, игр, корпоративных приложений и многого другого.

Преимущества Java представлены ниже.

1. Java программы могут выполняться на различных операционных системах без изменений, благодаря платформе Java Virtual Machine (JVM).

2. Java поставляется с обширной стандартной библиотекой, которая обеспечивает разработчиков готовыми решениями для ряда задач.

3. Java обеспечивает поддержку многопоточности, что делает его подходящим для создания многозадачных и высокопроизводительных приложений.

4. Java имеет встроенные механизмы безопасности, такие как проверка типов во время выполнения (bytecode verification) и система безопасности.

Недостатки Java представлены ниже.

1. Интерпретация и выполнение кода через виртуальную машину JVM может привести к некоторым накладным расходам и снижению производительности по сравнению с некоторыми низкоуровневыми языками.

2. Некоторые критики указывают на более высокое потребление памяти JVM в сравнении с некоторыми другими языками.

3. Для создания современных интерфейсов требуется использование сторонних библиотек, что может усложнить разработку пользовательских интерфейсов.

NestJS

Это фреймворк для создания масштабируемых и эффективных серверных приложений на языке программирования TypeScript. NestJS основан на архитектуре модулей и использует принципы объектно-ориентированного программирования, функционального программирования и реактивного программирования. Он предоставляет разработчикам целый набор инструментов и функций для создания веб-приложений, микросервисов и других серверных приложений.

Некоторые ключевые особенности NestJS представлены ниже.

1. Использование TypeScript для статической типизации и улучшения производительности и безопасности кода.

2. Поддержка REST API, GraphQL и WebSocket.

3. Возможность интеграции с различными фреймворками и библиотеками.

4. Модульность и расширяемость. Приложение можно построить из отдельных модулей.

5. Встроенная поддержка тестирования.

NestJS пользуется популярностью в сообществе разработчиков благодаря своей гибкости, простоте использования и возможности создания мощных и масштабируемых приложений.

NestJS предоставляет разработчикам множество возможностей для создания высокопроизводительных и удобных в обслуживании приложений, используя принципы модульности, инъекции зависимостей и другие передовые практики разработки. Кроме того, NestJS предлагает удобный способ организации кода с помощью декораторов и провайдеров, что делает приложения более структурированными и понятными [24].

TypeORM

Это Object-Relational Mapper (ORM) для TypeScript и JavaScript (с поддержкой для Node.js и браузера). Он предоставляет возможность удобно работать с реляционными базами данных, используя объектно-ориентированный подход, что позволяет разработчикам взаимодействовать с базой данных через объекты и классы, а не непосредственно через SQL-запросы.

TypeORM поддерживает различные типы баз данных, такие как PostgreSQL, MySQL, MariaDB, SQLite, MS SQL Server и другие.

TypeORM облегчает разработку приложений, особенно в среде TypeScript и Node.js, предоставляя удобные инструменты для работы с базами данных, без прямого использования SQL-запросов [27].

Вывод по первой главе

В процессе анализа веб-приложений и обзора и средства разработки было принято решение создавать веб-приложение, работающее с API Metamask и Yandex GPT.

Для реализации интерфейса будут использованы JavaScript, React Router(навигация), CSS-Modules/SCSS (стилизация), и Next.js (фреймворк React для создания динамических приложений).

Для функциональной части будут использованы TypeScript, Nest.js, PostgreSQL, Axios, TypeORM, UUID (Universally Unique Identifier), bcrypt (хэш-функция для защиты учетных данных).

2. ПРОЕКТИРОВАНИЕ

2.1. Требования к разрабатываемой системе

После рассмотрения предметной области и обзора доступных средств разработки были выявлены функциональные и нефункциональные требования, которые должны быть учтены при разработке веб-приложения.

В процессе проектирования приложения были выявлены следующие функциональные требования.

1. Система должна предоставлять возможность регистрации и авторизации данных пользователя.

2. Система должна иметь возможность подключения кошелька Metamask.

3. Система должна предоставлять возможность выхода из личного аккаунта пользователя.

4. Система должна предоставлять возможность приобретения подписки с помощью токенов.

5. Система должна иметь возможность отправки запроса с текстом.

6. Пользователь должен иметь возможность добавления, удаления запросов в истории сообщений.

7. Пользователь должен иметь возможность написать отзыв и поставить оценку.

При нажатии подключить кошелек, отправляется запрос к Metamask. Если у пользователя не установлено расширение Metamask в браузере, высветится надпись с просьбой установить расширение. Если у пользователя уже установлен Metamask, на экране появится окно для входа в криптокошелек, и он сможет войти, используя свои учетные данные.

В процессе проектирования приложения были выявлены следующие нефункциональные требования.

1. Веб-страница должна быть написана на HTML5 с использованием CSS3 и JavaScript (React.js).

2. Сайт должен корректно отображаться при запуске в популярных браузерах, таких как Яндекс.Браузер, GoogleChrome, Mozilla Firefox, Opera, Microsoft Edge.
3. Должен быть реализован адаптивный дизайн веб-приложения.
4. Система должна быть интегрирована с серверной частью.
5. Система должна работать с заранее созданным смарт контрактом.

2.2. Диаграмма вариантов использования

Для проектирования приложения был использован язык графического описания для объектного моделирования UML. По выдвинутым к системе требованиям, была составлена диаграмма вариантов использования. Диаграмма приведена на рисунке 14.

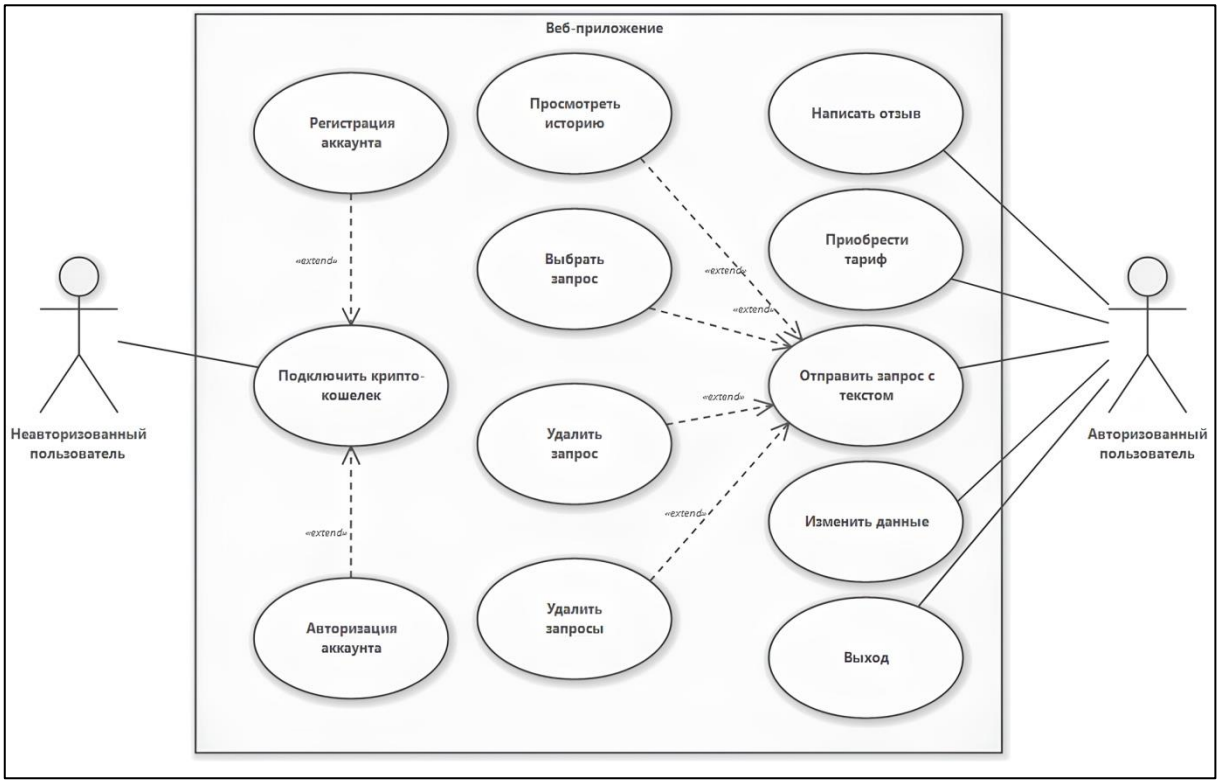

Рисунок 14 – Диаграмма вариантов использования веб-приложения

На диаграмме представлено два актера – авторизованный и неавторизованный пользователь веб-приложения.

Авторизованный пользователь – это человек, зарегистрированный и прошедший авторизацию на сайте, ему доступен основной функционал веб-приложения.

Неавторизованный пользователь – это любой незарегистрированный на сайте человек, он не имеет доступа к основному функционалу.

Краткое описание вариантов использования для актера «Неавторизованный пользователь» представлено ниже.

1. Подключить крипто-кошелек. Возможность подключить крипто кошелек в системе.

2. Регистрация аккаунта. Возможность зарегистрироваться в системе.

3. Авторизация аккаунта. Возможность авторизоваться в системе под своими учетными данными.

Краткое описание вариантов использования для актера «Авторизованный пользователь» представлено ниже.

1. Отправить запрос с текстом. Возможность отправки запроса текстового сообщения.

2. Выбрать запрос. Возможность выбора запроса текстового сообщения.

3. Удалить запрос. Возможность удаления запроса текстового сообщения.

4. Удалить запросы. Возможность удаления всех запросов текстового сообщения.

5. Просмотреть историю. Возможность просмотра истории ответов.

6. Приобрести тариф. Возможность приобретения тарифа, после использования лимита бесплатных попыток пользователя.

7. Написать отзыв. Возможность дать оценку и написать отзыв о приложении.

8. Выход. Возможность выхода из системы пользователем.

9. Изменить данные. Изменить учетные данные пользователя.

Спецификация диаграммы вариантов использования приведена в приложении А.

2.2. База данных веб-приложения

Как и для любого сайта, так и для данного веб-приложения требуется база данных. Для разработки веб-приложения используется PostgreSQL.

Это мощная система управления реляционными базами данных (СУБД), которая обеспечивает надежное хранение и обработку данных.

PostgreSQL полностью совместим с SQL, что обеспечивает простоту в использовании для разработчиков, знакомых с языком запросов.

Поддерживает создание пользовательских функций, типов данных и расширений, что позволяет разработчикам расширять функциональность базы данных по своим потребностям.

Известен своей высокой надежностью и стабильностью работы даже при очень больших объемах данных.

Поддерживает горизонтальное и вертикальное масштабирование, что позволяет эффективно работать с базами данных любого размера.

Поддерживает транзакции с использованием механизма ACID, обеспечивая целостность данных даже при параллельных операциях.

Поддерживает различные типы данных, включая географические и текстовые, а также поддерживает индексацию, полнотекстовый поиск и другие расширенные функции.

Эти особенности делают PostgreSQL одной из наиболее популярных и надежных систем управления базами данных в мире [23].

ER-диаграмма представляет собой графическую модель базы данных, которая иллюстрирует взаимосвязи между элементами. Она состоит из двух основных компонентов: сущностей и связей.

ER-диаграмма, представленная на рисунке 15, демонстрирует взаимосвязи между разделами базы данных, такими как:

1) пользователи (`user`);
2) группа запросов (`question_group`);
3) запросы (`question`);
4) токен (`tokens`);
5) отзывы (`review`);
6) положительные отзывы (`review_like`);
7) отрицательные отзывы (`review_dislike`).

База данных `user` связана с базой данных `question_group`, `question`, `review_like`, `review_dislike` через связь между атрибутами `id-userId`. Также `user` связана с базой данных `review`, через связь между атрибутами `id-authorId`.

База данных `question_group` и база данных `question`, связаны через связь атрибутов `id-threadId`.

База данных `review` связана с базой данных `review_like`, `review_dislike`, через связь между атрибутами `id-reviewId`.

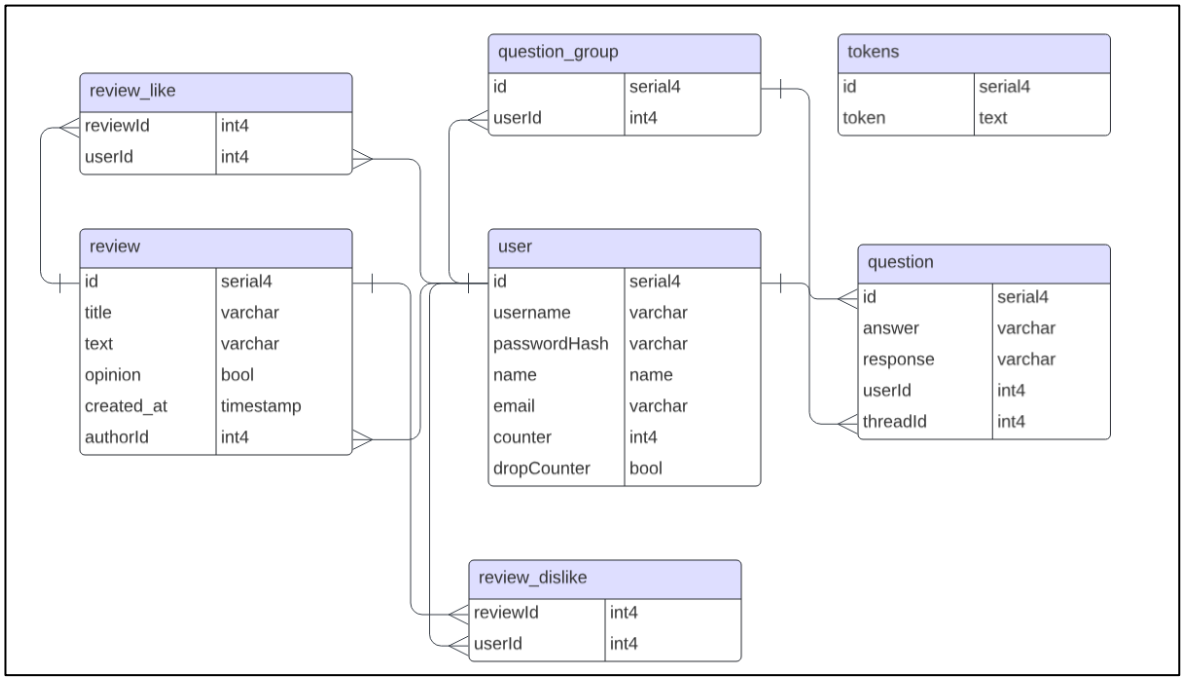

Рисунок 15 – Диаграмма базы данных

Тип `user` представляет информацию о пользователе, в котором добавлены дополнительные поля, такие как `username`, `name`,

passwordHash, email. Эти поля представляют различные атрибуты пользователя, такие как имя, имя пользователя, электронная почта, и хэшированный пароль.

Тип question_group представляет группу для сообщений, userId задает номер пользователя, который отправил запрос, а id задает номер запроса.

Тип question представляет информацию о запросе, в котором добавлены дополнительные поля answer, response, userId, threadId. Эти поля содержат информацию о запросе и ответе сообщений, номер пользователя, номер сообщения.

Тип tokens представляет информацию о токене, полученный с api, производимый после авторизации пользователя.

Тип review представляет информацию об отзывах, в котором добавлены дополнительные поля title, text, opinion, created_at, authorId. Эти поля содержат информацию о теме отзыва и тексте отзыва, оценка, время добавления, номер пользователя.

Тип review_like представляет группу для положительных отзывов, userId задает номер пользователя, который отправил положительный отзыв, а reviewId задает номер отзыва.

Тип review_dislike представляет группу для отрицательных отзывов, userId задает номер пользователя, который отправил положительный отзыв, а reviewId задает номер отзыва.

Вывод по второй главе

В процессе анализа требований были выведены основные функциональные и нефункциональные требования к разрабатываемому веб-приложению, на основе которых была составлена диаграмма вариантов использования и спецификация к ней. Также была описана база данных веб-приложения.

3. АРХИТЕКТУРА СИСТЕМЫ

3.1. Диаграмма деятельности

Для создания диаграммы деятельности был выбран прецедент «регистрация аккаунта» из диаграммы вариантов использования, представленной в пункте 2.2. На рисунке 16 представлена диаграмма деятельности для данного прецедента.

Неавторизованный пользователь заполняет регистрационную форму, затем нажимает кнопку «Регистрация». Если данные, введенные пользователем, некорректны, то появляется сообщение об ошибке регистрации.

Если же данные, введенные пользователем корректны, то формируется запрос к базе данных для добавления нового аккаунта, запрос исполняется и данные сохраняются.

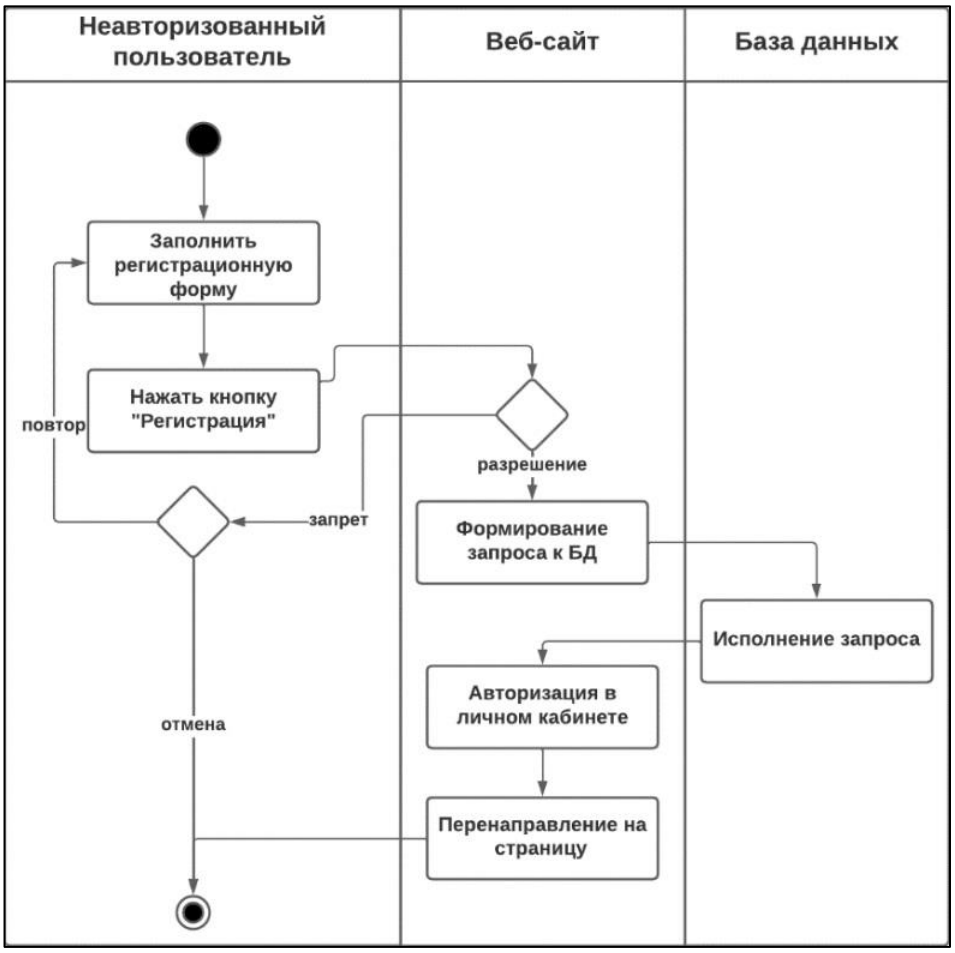

Рисунок 16 – Диаграмма деятельности для прецедента «регистрация аккаунта»

После успешной регистрации происходит автоматическая авторизация пользователя в личном кабинете, затем происходит перенаправление на главную страницу, где он может начать пользоваться функциональностью сайта, предоставляемой авторизованным пользователям.

Таким образом, процесс регистрации на сайте включает в себя заполнение формы, валидацию данных, сохранение аккаунта в базе данных, авторизацию пользователя и перенаправление на страницу с нейронной моделью.

3.2. Диаграмма компонентов

В качестве графического представления структуры разрабатываемого веб-приложения была спроектирована диаграмма компонентов, представленная на рисунке 17.

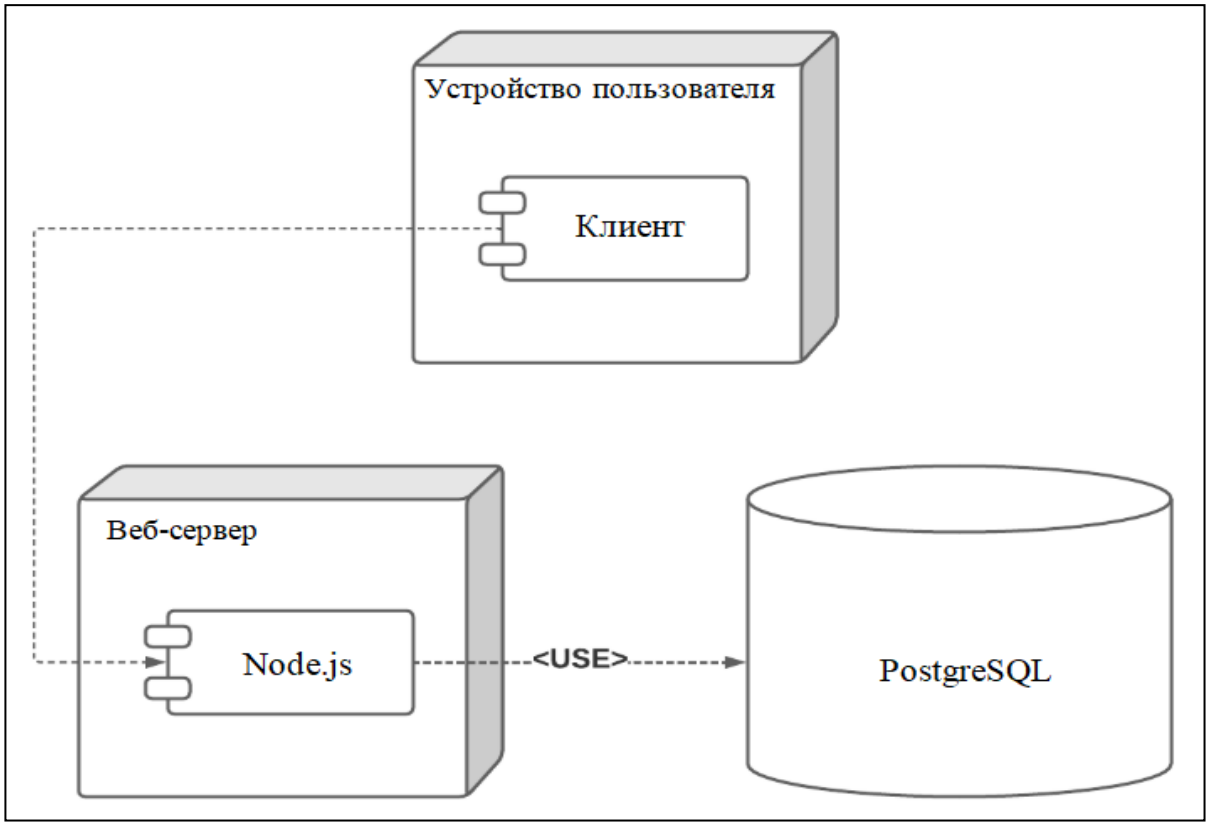

Рисунок 17 – Диаграмма компонентов

Слой представления (клиентская часть). Этот слой отвечает за представление пользовательского интерфейса и обрабатывает пользовательское взаимодействие с системой. Этот уровень базируется на визуальном представлении веб-приложения, обычно при разработке пользовательского интерфейса (UI) основным выбором индустрии является HTML в сочетании с JavaScript или такими фреймворками, как React, Angular, Ember.

Слой бизнес логики (веб-сервер). Этот слой отвечает за обработку и управление основной функциональности веб-приложения. Он получает запросы от слоя представления, выполняет необходимую обработку, взаимодействует с уровнем хранения данных и возвращает выходные данные на слой представления в пользовательский интерфейс, а также в базу данных. Он реализуется с помощью серверных языков программирования, в нашем случае применяется TypeScript совместно с Node.js

Слой хранения данных (база данных). Данный слой отвечает за хранение данных, которые используются при работе веб-приложения. Он может включать в себя базы данных, файловые системы или прочие механизмы хранения информации.

3.3. Диаграмма последовательности

Для создания диаграммы последовательности был выбран прецедент «приобрести тариф» из диаграммы вариантов использования, представленной в пункте 2.3. На рисунке 18 представлена диаграмма последовательности.

Авторизованный пользователь нажимает на кнопку «Приобрести доступ». После нажатия на кнопку, приложение обрабатывает запрос пользователя и отправляет в расширение Metamask.

Далее, у пользователя появилось окно с операцией отправки, необходимо подтвердить действие обработки смарт-контракта, если баланс

кошелька имеет нужное количество токенов, в расширение Metamask будет отображено сообщение, что токенов хватает для подтверждения операции и нажать на кнопку подтверждения.

После успешной отправки токенов, происходит передача данных об успешной сделке смарт-контракту, смарт-контракт подтверждает операцию Metamask, а Metamask, в свою очередь сообщает приложению об успешной сделке передачи токенов. После совершенной сделки, пользователь приобрел тариф и автоматически переходит на страницу с нейронной моделью с обновленным тарифом.

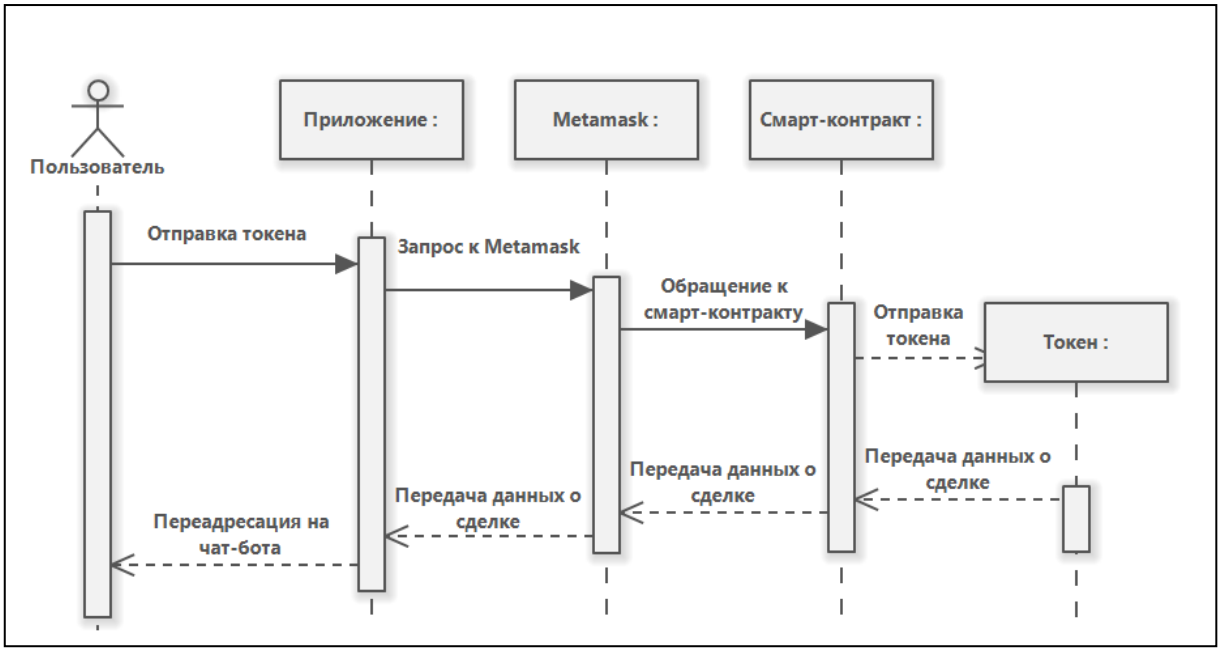

Рисунок 18 – Диаграмма последовательности для прецедента «Приобрести тариф»

Вывод по третьей главе

В рамках проектирования системы были разработаны диаграмма деятельности и диаграмма последовательности, описывающие поведение некоторых прецедентов из диаграммы вариантов использования, представленной в пункте 2.2, а также диаграмма компонентов архитектуры системы.

4. РЕАЛИЗАЦИЯ

4.1. Реализация функционала сервера

Для создания функционала сервера приложения был использован фреймворк Nest.js на языке TypeScript.

Реализация компонента `main.ts` представлена в листинге 1. Данный компонент в Nest.js представляет корневой компонент и является основным в точке входа сервера приложения.

Листинг 1 – Компонент `main.ts`

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors({
    origin: 'http://localhost:3000',
    credentials: true,
  });  app.use(cookieParser());
  await app.listen(4000);
}
async function reqBearer(){
  await axios({
    url: ' https://llm.api.cloud.yandex.net/foundationModels/v1/completion,
    method: 'POST',
    headers: {
      'Content-Type' : 'application/x-www-form-urlencoded',
      'RqUID' : uuidv4(),
    },
    auth: {
      username: '',
      password: '',
    },
    data: parseStringify({
      scope: 'YandexGPT_API'
    }),
    httpsAgent: new https.Agent({
      rejectUnauthorized: false,
    })
  }).then(async (res) => {
    await MainDatasource.getRepository(tokens).delete({
      token: Not(IsNull())
    })
    await MainDatasource.getRepository(tokens).save({
      token: res.data.access_token
    })
  })
}
bootstrap().then(res=>{
  console.log('Server starts.')
  MainDatasource.initialize().then(async ()=>{
    console.log('Postgres SQL DB init')
    await reqBearer()
    setInterval(async ()=>{
      await reqBearer()
    }, 1000 * 60 * 15)
  })
});
```

Код представляет собой асинхронную функцию, которая создает приложение Nest.js и подключает модуль приложения.

Содержит технологию CORS для приложения, разрешает запросы только с определенного источника и включает передачу учетных данных.

Использует программное обеспечение middleware, которое помогает приложению и серверу обмениваться друг с другом запросами. И запускает сервер, для обработки входящих запросов.

Далее обрабатывается функция `reqBearer`, которая отправляет POST запрос на URL с определенными заголовками, аутентификацией и данными для подключения . Затем сохраняется полученный токен доступа в базу данных.

Далее запускается функция `bootstrap` для настройки базы данных, вызывается функция `raqBearer`, после этого устанавливается интервал обновления токена каждые 15 минут.

Реализация компонента `app.module.ts`, необходима для подключения модулей, базы данных и установки внешних контроллеров и провайдеров, задействованные в сервере приложения.

Провайдеры обеспечивают доступ к определенным данным и функциональности внутри компонента, а контроллеры управляют логикой и взаимодействием с провайдерами.

Каждый компонент играет свою специфическую роль в инициализации сервера, управлении данными пользователя и обработке запросов к API. Код представлен в листинге 2.

Листинг 2 – Компонент `app.module.ts`

```
@Module({
  imports: [
      ConfigModule.forRoot({
          envFilePath: './backend.env'
      }),
      ChatModule,
      UserModule,
      ScheduleModule.forRoot(),
      HttpModule,
      HttpsModule,
      TypeOrmModule.forRoot({
        type: 'postgres',
```

```
        host: '85.192.48.192',
        port: 6543,
        username: 'postgres',
        password: 'TheMili7aryB0rderCond1t1on',
        database: 'postgres',
        entities: MainDatasourceOptions.entities,
        synchronize: true
      }),
  ],
  controllers: [AppController, UserController, ChatController,
ReviewController],
  providers: [AppService, UserService, ChatService, ReviewService],
})
export class AppModule {}
```

Этот код использует провайдеры и контроллеры для четырех компонентов: приложения, данных пользователя, отзывов и запросов.

В реализации компонента `chat.controller.ts`, разработаны методы для отправки сообщения в чат, удаления конкретной ветки чата, удаления всех веток чата и получения списка веток чата для конкретного пользователя. Код представлен в листинге 3.

Листинг 3 – Компонент `chat.controller.ts`

```
@Controller('chat')
export class ChatController {
    constructor (
        private chatService:ChatService
    ){}
    @Public()
    @Post('/')
    async getMessage(@Req() prompt: any) {
        return await this.chatService.sendChat(prompt)
    }
    @Delete('/dropThread')
    async dropThread(@Body() res: any, @Req() req: any) {
        return await this.chatService.dropThread(res?.threadId, req)
    }
    @Delete('/dropAllThreads')
    async dropAllThreads(@Req() req: any) {
        return await this.chatService.dropAllThreads(req)
    }
    @Get('/getUserThreads')
    async getUserThreads(@Req() req: any) {
        return await this.chatService.getUserThreads(req)
    }
}
```

В коде конструктор принимает сервис чата `(ChatService)` в качестве зависимости.

Метод `getMessage` обрабатывает `POST` запросы по пути `'/'`. Возвращает результат вызова метода `sendChat` из `ChatService`, который отправляет сообщение в чат.

Метод `dropThread` обрабатывает DELETE запросы по пути `'/dropThread'`. Принимает тело запроса и объект запроса. Вызывает метод `dropThread` из `ChatService` для удаления конкретной ветки чата по указанному идентификатору ветки.

Метод `dropAllThreads` обрабатывает DELETE запросы по пути `'/dropAllThreads'`. Принимает объект запроса. Вызывает метод `dropAllThreads` из `ChatService` для удаления всех веток чата для текущего пользователя.

Метод `getUserThreads` обрабатывает GET запросы по пути `'/getUserThreads'`. Принимает объект запроса. Вызывает метод `getUserThreads` из `ChatService` для получения списка веток чата для конкретного пользователя. Все методы помечены декораторами для указания типа запроса и пути.

Следующий компонент `chat.service.ts` подразделен на несколько функций.

Первая функция `sendChat`, разработанная в компоненте `chat.service.ts`, необходима для отправки запроса с текстом. Код функции представлен в листинге 4.

Листинг 4 – Функция `sendChat`

```
async sendChat(req: any) {
        const BearerToken = await
MainDatasource.getRepository(tokens).findOne({
            where: {
                token: Not(IsNull())
            }
        })
        let history:any[] = [{role:"user", content:req.body.prompt}]
        if (req.body?.threadId) {
            const thread = await
MainDatasource.getRepository(QuestionGroup).find({
                where:{
                    id: req.body?.threadId
                },
                relations: {
                    questions: true
```

```
                }
            })
        }
        return this.httpService.post(
'https://llm.api.cloud.yandex.net/foundationModels/v1/completion',
            {
                model: "gpt://*******/yandexgpt-lite",
                //@ts-ignore
                messages: history,
                temperature: 1,
                top_p: 0.1,
                n: 1,
                stream: false,
                max_tokens: 512,
                repetition_penalty: 1,
                update_interval: 0,
            },
            {
                method: "post",
                headers: {
                    'Authorization': `Bearer ${String(BearerToken.token)}`,
                    'Content-Type': `application/json`,
                },
                httpsAgent: new https.Agent({
                    rejectUnauthorized: false,
                })
            }
        ).pipe(
            map((response:any) => { //@ts-ignore
                this.checkAndSave(
                    req.body?.prompt,
                    response.data.choices[0].message.content,
                    req,
                    req.body?.threadId
                return {bot: response.data.choices[0].message.content}
```

В коде функции создается токен авторизации в API сервиса. После этого токен сохраняется, и формируется запрос. Если в запросе имеется номер ветки, то запрос сохраняется. После сохранения, отправляется POST запрос с данными для конкретной модели с заданными параметрами модели нейронной сети. Следующим шагом отправляется токен авторизации в формате запроса JSON, полученный из базы данных.

Далее, идет конструкция, отключающая проверку на SSL сертификаты, выводится ответ сообщения модели нейронной сети и сохраняет необходимую информацию для функции `checkAndSave`.

Функция `checkAndSave` служит для сохранения информации в базу данных. Код функции представлен в листинге 5.

Листинг 5 – Функция `checkAndSave`

```
async checkAndSave(response: string, answer: string, userId: any, threadId:
number) {
        const keyToken = userId.cookies['Bearer']
        const userDecode = this.jwtService.decode(keyToken)
        let qg = null
        if (!userDecode) return
        if (threadId) {
            qg = await
MainDatasource.getRepository(QuestionGroup).findOne({
                where: {
                    user: {
                        id: userDecode?.userId
                    }
                }
            })
        } else {
            qg = await
MainDatasource.getRepository(QuestionGroup).save(Object.assign(new
QuestionGroup(),{
                user: {
                    id:userDecode?.userId
                }
            }))
        }
        await MainDatasource.getRepository(Question).save(Object.assign(new
Question(),{
            response: response,
            answer: answer,
            user: userDecode?.userId,
            thread: {id: qg?.id},
        }))
    }
```

В коде функции создается токен авторизации для получения данных пользователя, декодируем через `JWT` сервис, далее функция проверяет на наличие ветки с объектом, если ветки не было найдено, то создается ветка с новым объектом. После этого добавляется еще один объект с вопросом.

Следующая функция `getUserThreads`, предназначена для получения пользовательских веток, для извлечения всех групп вопросов пользователя из базы данных. Код функции представлен в листинге 6.

Листинг 6 – Функция `getUserThreads`

```
async getUserThreads(request: any){
        const keyToken = request.cookies['Bearer']
        const userDecode = this.jwtService.decode(keyToken)
        return await MainDatasource.getRepository(QuestionGroup).find({
            where:{
                user: {
                    id: userDecode?.userId
                }
            },
            relations:{
```

```
                questions: true
            }
        })
    }
```

В коде функции извлекается токен доступа из `cookies` запроса и декодируется с помощью сервиса `JWT`, использует метод для получения репозитория сущности и вызывает метод для получения всех групп вопросов, принадлежащих пользователю, где `ID` пользователя должен соответствовать `ID` пользователя из декодированного токена. Также устанавливается связь с вопросами, чтобы получить данные о вопросах, принадлежащих каждой группе.

Следующая функция `dropThread` предназначена для удаления одной ветки запроса пользователя. Код функции представлен в листинге 7.

Листинг 7 – Функция `dropThread`

```
async dropThread(threadId: number, request: any) {
        const keyToken = request.cookies['Bearer']
        const userDecode = this.jwtService.decode(keyToken)
        await MainDatasource.getRepository(QuestionGroup).delete({
            id: threadId,
            user: {
                id: userDecode?.userId
            }
        })
        return MainDatasource.getRepository(QuestionGroup).find({
            where:{
                user: {
                    id: userDecode?.userId
                }
            },
            relations: {
                questions: true
            }
        })
    }
```

Этот код представляет собой асинхронную функцию, которая используется для удаления конкретного вопроса по его идентификатору. В функции доступ к запросу происходит через `request` и извлекается токен доступа из `cookies` с помощью ключа `'Bearer'`. Затем декодируется пользовательский идентификатор из токена и происходит удаление вопроса из базы данных.

Следующая функция `dropAllThread` предназначена для удаления всех веток запроса пользователя. Код функции представлен в листинге 8.

Листинг 8 – Функция `dropAllThread`

```
async dropAllThreads(request:any){
        const keyToken = request.cookies['Bearer']
        const userDecode = this.jwtService.decode(keyToken)
        await MainDatasource.getRepository(QuestionGroup).delete({
            user: {
                id: userDecode?.userId
            }
        })
    }
}
```

Этот код представляет асинхронный метод для удаления всех записей из таблицы, основываясь на пользователе, связанном с этими записями.

В реализации компонента `user.controller.ts`, разработаны методы для авторизации и регистрации, получение одного пользователя, получение всех пользователей и получения текущего пользователя. Код представлен в листинге 9.

Листинг 9 – Компонент `user.controller.ts`

```
export class UserController {
    constructor(
        private readonly userService:UserService
    ) {}
    @Public()
    @Post('login')
    signIn(@Body() signInDto: Record<string, any>, @Res({passthrough:true})
response: Response) {
        return this.userService.signIn(signInDto.username,
signInDto.password, response);
    }
    @Public()
    @Post('/register')
    registerUser(@Body() signInDto: User) {
        return this.userService.registerUser(signInDto);
    }
    @Post('getUser')
    getUser(@Body() user:User){
        return this.userService.findOne(user.username)
    }
    @Post('getUsers')
    getUsers(@Body() pagination: pagination){
        return this.userService.getUsers(pagination)
    }
    @Get('/getMe')
    getMe(@Req() request: any){
        return this.userService.getMe(request)
```

Конструктор класса `UserController` принимает параметр `userService` типа `UserService`.

Метод `signIn` помечен декораторами, что означает, что этот метод доступен публично и обрабатывает POST запрос на путь `'/login'`. Метод принимает объект `signInDto` и объект `response` типа `Response` и вызывает метод `signIn` сервиса `userService`, передавая ему данные для аутентификации.

Метод `registerUser` также помечен декораторами и обрабатывает POST запрос на путь `'/register'`. Метод принимает объект `signInDto` типа `User` и вызывает метод `registerUser` сервиса `userService` для регистрации нового пользователя.

Метод `getUser` помечен декоратором и обрабатывает POST запрос на путь `'/getUser'`. Метод принимает объект `user` типа `User` и вызывает метод `findOne` сервиса `userService` для поиска пользователя по имени пользователя.

Метод `getUsers` помечен декоратором и обрабатывает POST запрос на путь `'/getUsers'`. Метод принимает объект `pagination` и вызывает метод `getUsers` сервиса `userService` для получения списка пользователей с учетом пагинации.

Метод `getMe` помечен декоратором и обрабатывает GET запрос на путь `'/getMe'`. Метод принимает объект `request` и вызывает метод `getMe` сервиса `userService` для получения информации о текущем пользователе.

Следующий компонент `user.service.ts` подразделен на несколько функций.

Первая функция `signIn`, разработанная в компоненте `user.service.ts`, необходима для авторизации пользователя в веб-приложении. Код функции представлен в листинге 10.

Листинг 10 – Функция `signIn`

```
async signIn(username: string, pass:string, response: any) {
    if (!(username && pass)) return {
```

```
            type: "ERROR",
            massage: "Введите данные для входа"
        }
        const user = await MainDatasource
            .createQueryBuilder()
            .from(User, "user")
            .where("user.username =:username", {username: username})
            .select("user")
            .addSelect("user.passwordHash")
            .getOne()
        if (!user){
            return {
                type: 'ERROR',
                massage: 'Указанный пользователь не существует'
            }
        } else {
            const isMatch = await bcrypt.compare(pass, user.passwordHash);
            if (!isMatch){
                throw new UnauthorizedException();
            } else {
                const payload = { userId: user.id, username: user.username };
                const token = await this.jwtService.signAsync(payload)
                delete user.passwordHash
                response.cookie('Bearer', token)
                return {
                    user: user,
                    access_token: token,
                };
            }
        }
}
```

Код сначала проверяет, были ли переданы имя пользователя и пароль. Если один из них отсутствует, возвращается сообщение об ошибке.

Затем код использует базу данных и `User` сущность, чтобы найти пользователя по имени пользователя. Извлекает хэш пароля пользователя, чтобы сравнить его с переданным паролем.

После этого код использует `bcrypt` для сравнения переданного пароля с хэшем пароля пользователя. Если пароли совпадают, пользователь успешно авторизован. Если пароли не совпадают, возвращается сообщение об ошибке авторизации. Если пароли не совпадают, то код выбрасывает исключение `UnauthorizedException`, что означает, что пользователь не авторизован.

Если пароли совпадают, то генерируется `JWT` токен с `payload`, содержащим `id` и имя пользователя. Затем пользовательский хэш пароля удаляется из объекта пользователя, и токен записывается в `cookies` под

именем 'Bearer'. Наконец, возвращается объект с данными пользователя и сгенерированным токеном.

Следующая функция registerUser, разработанная в компоненте user.service.ts, необходима для регистрации пользователя в веб-приложении. Код функции представлен в листинге 11.

Листинг 11 – Функция registerUser

```
async registerUser(user:User){
        // Проверка на человека с таким же Username
        const checkUser = await
MainDatasource.getRepository(User).findOneBy({
            username: user.username
        })
        if (checkUser === null){
            const passwordSalt = await bcrypt.genSalt();
            console.log(passwordSalt, user.passwordHash)
            user.passwordHash = await bcrypt.hash(user.passwordHash,
passwordSalt);
            const regUser = new User()
            regUser.username = user.username
            regUser.passwordHash = user.passwordHash
            regUser.name = user.name
            regUser.email = user.email
            await
MainDatasource.manager.save(regUser).then((res)=>{}).catch((e)=>{console.lo
g(e)})
            delete regUser.passwordHash
            return regUser
        } else {
            return {
                type: 'ERROR',
                massage: 'Данный пользователь уже существует'
            }
        }
    }
```

Код функции проверяет, есть ли уже пользователь с таким же именем пользователя. Для этого используется запрос к базе данных. Результат проверки сохраняется в переменной checkUser.

Если пользователь с таким именем пользователя не найден, то генерируется соль для хэширования пароля, затем пароль пользователя хэшируется с использованием bcrypt.hash. Хэшированный пароль сохраняется обратно в поле user.passwordHash.

Создается новый экземпляр пользователя regUser, заполняется информацией о пользователе (имя, почта, имя пользователя и

хэшированный пароль). После этого новый пользователь сохраняется в базе данных. В случае успешного сохранения пользователь возвращается как результат функции. Если пользователь с таким именем пользователя уже существует, то возвращается объект с информацией об ошибке, указывающий, что такой пользователь уже существует.

Таким образом, код выполняет проверку наличия пользователя с определенным именем пользователя в базе данных, и, если такого пользователя нет, выполняет регистрацию нового пользователя с хэшированным паролем.

Следующая функция `getUsers`, разработанная в компоненте `user.service.ts`, необходима для получения списка всех пользователей. Код функции представлен в листинге 12.

Листинг 12 – Функция `getUsers`

```
async getUsers(pagination: pagination){
        return  await MainDatasource.getRepository(User)
            .createQueryBuilder("user")
            .leftJoinAndSelect("user.roles", "role")
            .skip(pagination.page * pagination.rowsPerPage)
            .take(pagination.rowsPerPage)
            .getMany()
    }
```

Код представляет функцию, которая принимает параметр `pagination` и возвращает список пользователей в соответствии с пагинацией.

Внутри функции используется метод `getRepository` класса `MainDatasource` для получения репозитория, связанного с сущностью `User`. Затем создается запрос с помощью метода `createQueryBuilder`, указывается основная сущность как `user` и выполняется объединение с сущностью ролей.

Далее применяется пагинация, с помощью методов `skip`, для пропуска определенного количества записей и `take` для выбора определенного количества записей. Пагинация осуществляется на основе

параметров из объекта pagination, где page указывает на номер страницы, а rows-PerPage указывает на количество записей на странице.

Затем вызывается метод getMany для выполнения запроса и получения списка пользователей с их ролями. Полученные данные возвращаются из функции.

Следующая функция getMe, разработанная в компоненте user.service.ts, необходима для получения текущего пользователя из базы данных. Код функции представлен в листинге 13.

Листинг 13 – Функция getMe

```
async getMe(request:any){
        let res = undefined
        if (request.res.req.user){
            res =  await MainDatasource.getRepository(User)
                .createQueryBuilder("user")
                .where(`user.id = ${request.res.req.user.userId}`)
                .getOne()
        }
        return res
    }
}
```

Этот код представляет асинхронную функцию getMe, которая принимает запрос request в качестве параметра. Функция проверяет, есть ли пользователь в запросе, и если он есть, то выполняет запрос к базе данных, чтобы получить объект пользователя.

Следующая функция updateMe, разработанная в компоненте user.service.ts, необходима для обновления данных пользователя из базы данных. Код функции представлен в листинге 14.

Листинг 14 – Функция getMe

```
async updateMe(request: any) {
        const token = request.cookies['Bearer'];
        const decodeUser = this.jwtService.decode(token)
        if (['passwordHash',
'username'].map(x=>request.body.user.hasOwnProperty(x)).find(x=>x)) {
            throw new HttpException('В объект передаются защищенные поля
username или passwordHash', HttpStatus.FORBIDDEN)
        }
        const reqUser = await MainDatasource.getRepository(User).findOne({
            where: {
                id: decodeUser.userId
            }
        })
```

```
        return await MainDatasource.getRepository(User).save(Object.assign(
            reqUser, request.body.user
        ))
    }
```

Этот код обновляет пользователя, исключая защищенные поля `username` и `passwordHash` из запроса. Если в запросе передаются эти защищенные поля, будет сгенерировано исключение. После этого данные пользователя обновляются и сохраняются в базе данных.

Следующая функция `dropCounter`, разработанная в компоненте `user.service.ts`, необходима для снятия ограничения попыток ввода текста. Код функции представлен в листинге 15.

Листинг 15 – Функция `dropCounter`

```
async dropCounter(request: any) {
        const token = request.cookies['Bearer'];
        const decodeUser = this.jwtService.decode(token)
        if (decodeUser.userId) {
            await MainDatasource.getRepository(User).update({
                id: decodeUser.userId
            }, {
                dropCounter: true
            })
        } else {
            throw new HttpException('Не получатеся получить пользователя',
HttpStatus.NOT_FOUND)
        }
    }
```

Этот код представляет асинхронную функцию `dropCounter`, которая принимает запрос `request` в качестве аргумента. В этой функции извлекается токен доступа из `cookie`, далее декодируется пользователь из этого токена.

После того, как пользователь успешно декодирован, происходит обновление записи в базе данных. В данном случае используется метод `getRepository` из класса `MainDatasource` для получения репозитория для сущности `User`. Затем выполняется обновление записи в базе данных для пользователя с заданным идентификатором, устанавливая значение `dropCounter` в `true`.

Если не удается получить пользователя из токена или декодированный пользователь не содержит идентификатор, то выбрасывается исключение.

При вызове функции `addReview`, выполняется обработчик для добавления отзыва, который написал пользователь. Код функции представлен в листинге 16.

Листинг 16 – Функция `addReview`

```
async addReview(review: Review, req: any) {
        const token = req.cookies['Bearer'];
        const decodeUser = this.jwtService.decode(token)
        const reqUser = await MainDatasource.getRepository(User).findOne({
            where: {
                id: decodeUser.userId
            }
        })
        const reviewCheck = await
MainDatasource.getRepository(Review).findOne({
            where:{
                author: {
                    id: reqUser.id
                }
            }
        })
        if (reviewCheck) {
            throw new HttpException('Вы уже написали отзыв',
HttpStatus.FORBIDDEN)
        } else {
            review.author = reqUser
            return await
MainDatasource.getRepository(Review).save(Object.assign(
                new Review(), review
            ))
        }
    }
```

Этот код добавляет отзыв от пользователя. Сначала извлекается токен из `cookie` запроса. После этого токен декодируется для получения данных пользователя.

Далее выполняется поиск пользователя в базе данных по `ID`, полученному из декодированного токена. Затем выполняется проверка наличия отзыва от этого пользователя в базе данных.

Если отзыв уже существует, генерируется исключение с сообщением "Вы уже написали отзыв". Если отзыв отсутствует, пользователь присваивается отзыву и отзыв сохраняется в базе данных.

Таким образом, код проверяет, есть ли уже отзыв от пользователя в базе данных, и если нет, добавляет новый отзыв от этого пользователя.

При вызове функции `addOpinion`, выполняется обработчик для добавления мнения отзыва, который написал пользователь. Код функции представлен в листинге 17.

Листинг 17– Функция `addOpinion`

```
async addOpinion(review: Review, req: any, opinion: string) {
        const token = req.cookies['Bearer'];
        const decodeUser = this.jwtService.decode(token)
        const reqUser = await MainDatasource.getRepository(User).findOne({
            where: {
                id: decodeUser.userId
            }
        })
        const reviewCheck = await
MainDatasource.getRepository(Review).findOne({
            where:{
                author: {
                    id: reqUser.id
                },
            },
            relations: {
                dislikedUsers: true,
                likedUsers: true
            }
        })
        if (reviewCheck && reqUser) {
            if (
                reviewCheck.likedUsers.map(x=>x.id).includes(reqUser.id) ||
                reviewCheck.dislikedUsers.map(x=>x.id).includes(reqUser.id)
            ) {
                reviewCheck.dislikedUsers =
reviewCheck.dislikedUsers.filter(x=>x.id !== reqUser.id)
                reviewCheck.likedUsers =
reviewCheck.likedUsers.filter(x=>x.id !== reqUser.id)
            }
            if (opinion === 'like') reviewCheck.likedUsers =
reviewCheck.likedUsers.concat(reqUser)
            else if (opinion === 'dislike') reviewCheck.dislikedUsers =
reviewCheck.dislikedUsers.concat(reqUser)
            await MainDatasource.getRepository(Review).save(reviewCheck)
            return this.getReview(reviewCheck.id)
        } else {
            throw new HttpException('Не найден пользователь или отзыв',
HttpStatus.NOT_FOUND)
        }
    }
```

Этот код представляет собой метод `addOpinion`, который добавляет положительное или отрицательное мнение пользователя к отзыву.

Извлекается токен из cookies запроса.

Токен декодируется с использованием сервиса JWT для получения информации о пользователе.

Выполняется поиск пользователя в базе данных по идентификатору, извлеченному из расшифрованного токена.

Выполняется поиск отзыва, написанного данным пользователем, с загрузкой информации о пользователях, которым понравился или не понравился отзыв.

Если отзыв и пользователь найдены, происходит проверка наличия мнения пользователя о данном отзыве. Если мнение уже существует, тогда мнение удаляется.

В зависимости от переданного мнения, либо добавляется пользователь в список положительных отзывов, либо в список отрицательных. Обновленная информация об отзыве сохраняется в базе данных.

Возвращается обновленная информация об отзыве с помощью функции this.getReview(reviewCheck.id).

Если пользователь или отзыв не найдены, выбрасывается исключение с кодом состояния 404.

Этот код обрабатывает добавление мнения пользователя к отзыву, учитывая лайки и дизлайки, и сохраняет обновленные данные в базе данных.

В реализации компонента user.module.ts, разработаны методы контроллеры и middleware, необходимые для работы с пользователями в приложении. Код представлен в листинге 18.

Листинг 18 – Компонент user.module.ts

```
@Module({
    imports: [
        JwtModule.register({
            global: true,
            secret: jwtConstants,
            signOptions: {expiresIn: '7d'}
        })
    ],
    controllers: [UserController],
    providers: [
```

```
        UserService,
        {
            provide: APP_GUARD,
            useClass: AuthGuard
        }
    ]
})
```

Импортируется `JwtModule` и настраивается его использование в приложении. В данном случае он глобальный, который использует константу `jwtConstants` в качестве секрета для подписи токенов и устанавливает время жизни токена на 7 дней.

В разделе `controllers` указывается, что в модуле используется контроллер `UserController`. В разделе `providers` указываются провайдеры для использования в модуле. В данном случае это `UserService` для работы с пользователями и `AuthGuard` в качестве `middleware` для проверки авторизации.

Также в разделе `providers` устанавливается провайдер с использованием класса `AuthGuard`. Это позволяет использовать `AuthGuard` в качестве глобального `middleware`, для защиты всех маршрутов в приложении.

Код компонента `auth.guard.ts`, представляет собой реализацию класса `Guard` для аутентификации в NestJS. Данные классы используются для защиты маршрутов от доступа неавторизованных пользователей. Код представлен в листинге 19.

Листинг 19 – Компонент `auth.guard.ts`

```
export class AuthGuard implements CanActivate {
    constructor(private jwtService: JwtService, private reflector:
Reflector) {}

    async canActivate(context: ExecutionContext): Promise<boolean> {
        const isPublic =
this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
            context.getHandler(),
            context.getClass(),
        ]);
        if (isPublic) {
            return true;
        }

        const request = context.switchToHttp().getRequest();
```

```
        const token = request.cookies['Bearer'];
        if (!token) {
            throw new UnauthorizedException();
        }
        try {
            request['user'] = await this.jwtService.verifyAsync(token, {
                secret: jwtConstants,
            });
        } catch {
            throw new UnauthorizedException();
        }
        return true;
    }

    private extractTokenFromHeader(request: Request): string | undefined {
        const [type, token] = request.headers.authorization?.split(' ') ??
[];
        return type === 'Bearer' ? token : undefined;
    }
}
```

Конструктор класса `AuthGuard` принимает два параметра: `jwtService` и `reflector`. `jwtService` используется для работы с JWT токенами, а `reflector` используется для доступа к метаданным контроллера и его обработчика.

Метод `canActivate` является основным методом класса `AuthGuard`, который выполняет проверку доступа пользователя перед выполнением защищенного действия.

Вначале метод проверяет аннотацию с помощью `reflector`, чтобы определить, является ли маршрут открытым для публичного доступа. Если маршрут помечен как открытый, возвращается `true`.

Далее извлекается `JWT` токен из `cookies` запроса. Если токен не найден, выбрасывается исключение `UnauthorizedException`. Затем происходит верификация `JWT` токена. Если верификация успешна, в запрос добавляется объект пользователя, извлеченный из расшифрованного `JWT` токена. В случае возникновения ошибки при верификации токена также выбрасывается исключение `UnauthorizedException`.

Наконец, метод возвращает значение `true`, если проверка аутентификации успешно пройдена, что позволяет продолжить выполнение защищенного действия.

4.2. Реализация функционала блокчейна

При нажатии на кнопку «Подключить кошелек» вызывается функция, которая обращается к API Metamask [4]. На листинге 20 представлен фрагмент кода на языке JavaScript(React), описывающий функцию `connectWallet`.

Листинг 20 – Функция `connectWallet`

```
export const connectWallet = async () => {
  try {
    if (!window.ethereum) return alert("Установите Metamask расширение");
    const accounts = await window.ethereum.request({
      method: "eth_requestAccounts",
    });
    const firstAccount = accounts[0];
    return firstAccount;
  } catch (error) {
    console.log(error);
  }
};
```

Эта функция осуществляет подключение к кошельку Metamask. Когда пользователь вводит пароль от кошелька, программа берет имя кошелька и выводит его на экран.

После подключения аккаунта Metamask, вызывается функция `checkWallet`. Код функции представлен на листинге 21.

Листинг 21 – Функция `checkWallet`

```
export const ChechIfWalletConnected = async () => {
  try {
    if (!window.ethereum) return alert("Установите Metamask расширение");

    const accounts = await window.ethereum.request({
      method: "eth_accounts",
    });
    const firstAccount = accounts[0];
    return firstAccount;
  } catch (error) {
    alert('Авторизация уже в процессе, введите данные');
    console.log(error);
  }
};
```

При вызове функции, приложение проверяет, подключен ли кошелек к системе, в случае если кошелек не подключен, на экран выводится сообщение с просьбой установить расширение Metamask. Если

пользователь подключил расширение, но не авторизовался в крипто-кошелек, на экран выводится сообщение о том, что авторизация уже в процессе и необходимо ввести данные.

Если крипто-кошелек авторизован – система обращается к заранее созданному смарт-контракту и ABI этого контракта (способ работы с данными контракта). В параметры транзакции поступают заранее созданный контракт, запрос от приложения к системе Metamask. Далее с этими данными поступает запрос к Metamask, после чего пользователь может приобрести подписку.

После подключения аккаунта Metamask, вызывается функция `connectingWithContract`. Код функции представлен на листинге 22.

Листинг 22 – Функция `connectingWithContract`

```
export const connectingWithContract = async () => {
    try {
        const web3modal = new Web3Modal();
        const connection = await web3modal.connect();
        const provider = new ethers.providers.Web3Provider(connection);
        const signer = provider.getSigner();
        const contract = fetchContract(signer);
        return contract;
    } catch (error) {
        console.log(error);
    }
}
```

При вызове функции, приложение выполняет подключение к смарт-контракту, если крипто-кошелек подключен – система обращается к заранее созданному смарт-контракту и ABI этого контракта (способ работы с данными контракта). В параметры транзакции поступают заранее созданный контракт, запрос от приложения к системе Metamask. Далее с этими данными поступает запрос к Metamask, после чего пользователь может приобрести подписку.

После того, как пользователь выбрал тариф, вызывается функция `buyMembership`. Код функции представлен на листинге 23.

Листинг 23 – Функция `buyMembership`

```
const buyMembership = async (memberShip_id) => {
        const contract = await connectingWithContract();
```

```
    const connectAccount = await connectWallet();
    setAddress(connectAccount);

    try {
        if(memberShip_id) {
            const today = Date.now() + 2678400000;
            let date = new Date(today);
            const expiredDate = data.toLocalDateString('en-US');
            const money = ethers.utils.parseEther('1')

            const mintTransaction = await contract.mint(
                memberShip_id,
                connectAccount,
                expiredDate.toString(),
                {
                    value: money.toString(),
                }
            );
            await mintTransaction.wait();
            const freeTrail = JSON.stringify('Pro Member');
            localStorage.setItem('freeTrail', freeTrail);
            console.log('Taken membership', mintTransation);
            window.location.reload();
         }
    } catch (error) {
        console.log(error);
    }
};
```

Приложение запускает процесс приобретения тарифа, после подключения аккаунта Metamask, выполняется обработка покупки тарифа, задействуется созданный смарт-контракт и происходит выполнение транкзации токенов.

Начинает задействоваться смарт контракт, для покупки тарифа, и затем полностью выполняется транкзация токенов, как только смарт-контракт успешно будет совершен, пользователь автоматически перейдет на раздел с нейронной моделью с премиальным тарифом.

4.3. Реализация интерфейса веб-приложения

Веб-интерфейс был реализован с помощью CSS3 и HTML тэгов в React коде. Так как в React теги идут друг за другом и соответственным образом передаются props (элемент в React, для передачи данных между компонентами), было решено создать главный компонент `Sidebar`, в который входят остальные компоненты интерфейса (листинг 24).

Листинг 24 – Фрагмент кода компонента `Sidebar`

```
const SideBar = () => {
  return (
    <nav className='navbar navbar-expand-md p-0'>
      <button
       className='navbar-toggler d-none'
       type='button'
       data-bs-toggle="collapse"
       data-bs-target="#mainnavbarNav"
       aria-controls="mainnavbarNav"
       aria-expanded="false"
       aria-label="Toggle navigation"
       >
        <BiMenu className='mobil_custom_menu' />
       </button>
       <div className='collapse navbar-collapse' id='mainnavbarNav'>
       <div className='menu-panel'>
        <button
         className='mainnav-close d-block d-md-none'
         data-bs-toggle='collapse'
         data-bs-target='#mainnavbarNav'
        >
          <MdClose className='icon-custom' />
          <a href='/' className='logo-icon d-none d-md-flex'>
            <img src="assets/svg/logo-icon.svg" className='img-fluid'
alt="" />
          </a>
          <ul
            className='nav nav-tabs menu-wrapper'
            id="myTab"
            role="tablist"
          >
            <li className='nav-item'
            data-bs-toggle='tooltip'
            data-bs-placement='right'
            data-bs-title='chat'
            role='presentation'
          >
              <button
          className='nav-link active'
          type='button'
          data-bs-toggle="tab"
          data-bs-target="#chat"
          role='tab'
          aria-controls='chat'
          aria-selected='true'
```

Интерфейс приложения содержит список. Данный компонент предназначен для выбора пункта, с помощью которого пользователь может выбрать необходимый ему раздел.

В первом разделе отображена форма, в которой будут появляться история запросов, после отправки текста, а также кнопка удаления запроса по одному, и кнопка удаления всех запросов. Фрагмент кода представлен в листинге 25.

Листинг 25 – Фрагмент кода формы для истории запросов

```
<div class="main-wrapper">
        <nav class="navbar navbar-expand-lg bg-light p-0">
          <button
            class="navbar-toggler d-none"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarNav"
            aria-controls="navbarNav"
            aria-expanded="false"
            aria-label="Toggle navigation"
          >
            <BiMenu className="mobil_custom_menu" />
          </button>
          <div class="collapse navbar-collapse" id="navbarNav">
            <div class="inner-menu-panel">
              <button
                data-bs-toggle="collapse"
                data-bs-target="#navbarNav"
                class="inner-close d-block d-lg-none"
              >
                Назад
              </button>
              <div class="d-flex align-items-center gap-2">
            <h3 style={titleStyle}>Все чаты</h3>
          </div>
              <a data-cursor="pointer" class="btn-arrow" onClick={() =>
handleDeleteAll()}>
                Удалить чаты
                <div class="icon-arrow">
            </div>
          </a>
```

После отправки текста появляется запись с историей запроса. В этом объекте разработана кнопка для удаления запроса сообщения, отправленного нейронной модели. Фрагмент кода представлен в листинге 26.

Листинг 26 – Фрагмент кода записи истории запроса

```
<ul class="inner-links-list" id="innerLink">
{history.length > 0 && history.map((item, index) => (
  <li className="history-main" key={index} style={{flexDirection: "row",
position: "relative"}}>
    <div onClick={() => handleItemClick(item)} className="history-detail
text-truncate" style={{width: 'max-content'}}>
      <i className="iconsax" data-icon="message-text"></i>
      <div style={{width: 'max-content'}}>
        <p>{item.questions[item.questions.length - 1].response}</p>
      </div>
    </div>
    <a style={{width: 'max-content', position: "absolute", top:"0",
right:"0"}} data-cursor="pointer" className="btn-arrow" href=""
onClick={(e) => handleDeleteOne(e, item.id)}>
      <div className="icon-arrow">
```

```
      </div>
    </a>
  </li>
))}
</ul>
```

Также в разделе разработано формы для написания текста. Код представлен в листинге 27.

Листинг 27 – Отображение формы для написания текста

```
<form id="form_input_data" class="msger-inputarea ">
        <>
          <button
            class="navbar-toggler d-lg-none d-block msger-send-btn"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarNav"
            aria-controls="navbarNav"
            aria-expanded="false"
            aria-label="Toggle navigation"
          >
            <BiMenu className="icon_size" />
          </button>
          <input
            name="prompt"
            type="text"
            class="msger-input"
            placeholder="Задать вопрос..."
            rows="1"
            cols="1"
          />
          <button
            onClick={(e) => close(e)}
            type="submit"
            class="msger-send-btn"
          >
            <MdSend className="icon_size" />
          </button>
        </>
        </form>
```

В следующем разделе можем посмотреть историю ответов нейронной модели. Код представлен в листинге 28.

Листинг 28 – Отображение истории ответов чат-бота

```
<div class="main-section">
          <div class="container card p-0">
            <div class="card-header">
              <h3 class="text-white">Детальная история</h3>

              <form class="auth-form d-none d-md-block">
                <div class="form-group">
                  <i class="iconsax" data-icon="search-normal-2"></i>
                </div>
              </form>
            </div>
```

```
            <div class="card-body px-sm-4 px-3">
              <ul class="history-sec">
              {history.map((item, index) => (
                  <li className="history-main" key={index}>
                    <div  className="history-detail text-truncate">
                      <i className="iconsax" data-icon="message-text"></i>
                      <div>
                        <p>{item.questions[item.questions.length -
1].answer}</p>
                      </div>
                    </div>
                    <div className="history-time d-sm-flex d-none">
                      <ul>
                        <li>Чат</li>
                      </ul>
                    </div>
                  </li>
                ))}
              </ul>
            </div>
          </div>
```

В разделе отзывы имеется модальное окно с добавлением отзыва пользователя. В модальном окне отображены формы заполнения инфорации отзыва, оценка и кнопка добавления отзыва. Код фрагмента представлен в листинге 29.

Листинг 29 – Код фрагмента окна отзыва

```
<div class="modal rating-modal fade" id="staticBackdrop">
      <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
          <div class="modal-header">
            <h1 class="modal-title fs-5" id="staticBackdropLabel">
              Оцените работу приложения и напишите отзыв
            </h1>
            <button
              type="button"
              class="btn-close"
              data-bs-dismiss="modal"
              aria-label="Close"
            >
            </button>
          </div>
          <div class="modal-body">
            <p></p>
            <div class="col-sm-6 col-12">
              <div class="mb-3">
                <label for="firstname" class="form-label">
                  Тема отзыва
                </label>
                <input
                  type="email"
                  class="msger-input"
                  id="firstname"
                  value={title}
                  onChange={e => setTitle(e.target.value)}
                />
              </div>
```

```
              <div class="mb-3">
                <label for="firstname" class="form-label">
                  Отзыв
                </label>
                <input
                  type="email"
                  class="msger-input"
                  id="firstname"
                  value={message}
                  onChange={e => setMessage(e.target.value)}
                />
```

В следующем разделе отображена таблица с выбором тарифа подписки. В колонке есть кнопка выбора плана тарифа и его название. Код представлен в листинге 30.

Листинг 30 – Код таблицы тарифов

```
div class="col-xl-4 col-md-6 col-12">
              <div class="card inner-card">
                <div class="card-header">
                  <img
                   src="assets/svg/pricing/yearly.svg"
                   class="img-fluid"
                   alt=""
                  />
                  <h4 class="text-white mb-0">Премиум</h4>
                 </div>
                <div class="card-body">
                  <ul>
                   <li>Интеллектуальный чат-бот с искусственным
интеллектом</li>
                   <li>Неограниченное количество ответов на вопросы</li>
                   <li>Нет необходимости продлевать</li>
                   </ul>
                   <button class="no-select-plan select-plan"
                   onClick={() => buyMembership()}>
                    <i class="iconsax" data-icon="thick-arrow-right"></i>
                    Выбрать
                   </button>
                  </div>
                 </div>
               </div>
```

Далее идет раздел настроек приложения. В настройках имеется окна с обновлением новых данных пользователя и пароля аккаунта. В первом окне отображены формы заполения данных и кнопки обновления и отмены действий. Код представлен в листинге 31.

Листинг 31 – Отображение окна редактирование данных

```
<div className='main-section d-flex gap-4 flex-column'>
        <div className='container card p-0'>
          <div className='card-header'>
            <h3 className='text-white'>My account</h3>
```

```
          </div>
          <div className='card-body px-sm-4 px-3'>
            <div className='my-account'>
              <div className='user-detail'></div>
              <div className='user-main'>
                <div className='user-profile'>
                  <img
                    className='img-fluid'
                    alt=''
                  />
                  <i className='iconsax' data-icon='camera'></i>
                </div>
                <div className='user-option'>
                  <h4>{user?.name}</h4>
                  <p>{user?.email}</p>
                </div>
              </div>
              <form className='msger-inputarea mb-0'>
                <div className='row'>
                  <div className='col-sm-6 col-12'>
                    <div className='mb-3'>
                      <label>
                      </label>
                      <input
                      type="email"
                      id='firstname'
                      placeholder={user?.name}
                      onChange={(e) => handleFormFieldChange('name, e')}
                      className="msger-input"
                       />
                    </div>
                  </div>
                  <div className='col-sm-6 col-12'>
                    <div className='mb-3'>
                      <input
                      type="email"
                      id='lastname'
                      placeholder={user?.name}
                      onChange={(e) => handleFormFieldChange('surname, e')}
                      className="msger-input"
                       />
                    </div>
                  </div>
```

Во втором также отображены формы обновления нового пароля пользователя, а также кнопки обновления и отмены действий. Код представлен в листинге 32.

Листинг 32 – Отображение окна редактирования пароля

```
<div className='container card p-0'>
          <div className='card-header'>
            <h3 className='text-white'>Change password</h3>
          </div>
          <div className='card-body px-sm-4 px-3'>
          <div className='my-account'>
              <form className='msger-inputarea mb-0'>
                <div className='row'>
                  <div className='col-sm-6 col-12'>
                    <div className='mb-3'>
```

```
                      <label for='firstname' className='form-label'>
                        New password
                      </label>
                      <input
                      type="email"
                      id='firstname'
                      placeholder={' new password'}
                      onChange={(e) => handleFormFieldChange('password,
e')}
                      className="msger-input"
                       />
                    </div>
                  </div>
                  <div className='col-sm-6 col-12'>
                    <div className='mb-3'>
                      <label for='lastname' className='form-label'>
                        Confirm password
                      </label>
                      <input
                      type="email"
                      id='lastname'
                      placeholder={ 'Confirm password'}
                      onChange={(e) => handleFormFieldChange('name, e')}
                      className="msger-input"
                       />
                    </div>
                  </div>
                </div>
              </form>
            </div>
```

В разделе отображено окно, в котором есть 2 пункта выбора, регистрации и авторизации. Фрагмент кода пункта регистрации представлен в листинге 33.

Листинг 33 – Фрагмент кода с пунктом регистрации

```
div class="auth-form">
        <div class="mb-3 form-group">
          <i class="iconsax" data-icon="user-1"></i>
          <label for="name" class="form-label">
            Логин
          </label>
          <input
            type="name"
            placeholder="Введите логин"
            class="form-control"
            id="name"
            onChange={(e) => handleFormFieldChange("name", e)}
          />
        </div>
        <div class="mb-3 form-group">
          <i class="iconsax" data-icon="mail"></i>
          <label for="emailid" class="form-label">
            Email
          </label>
          <input
            type="email"
            placeholder="Введите email"
            class="form-control"
```

```
            id="emailid"
            onChange={(e) => handleFormFieldChange("email", e)}
          />
        </div>
        <div class="mb-3 form-group">
          <i class="iconsax" data-icon="lock-2"></i>
          <label for="password" class="form-label">
            Пароль
          </label>
          <input
            placeholder="Введите пароль"
            type="password"
            class="form-control"
            id="password"
            onChange={(e) => handleFormFieldChange("password", e)}
          />
        </div>
        <div class="mb-3 form-group">
          <i class="iconsax" data-icon="lock-2"></i>
          <label for="password1" class="form-label">
            Подтвердить пароль
          </label>
          <input
            placeholder="Введите пароль"
            type="password"
            class="form-control"
            id="password1"
            onChange={(e) => handleFormFieldChange("passwordConfirm", e)}
          />
        </div>
        <a
          class="btn-solid w-100 text-center mt-4"
          onClick={(e) => handleSignUp(e)}
        >
          Зарегистрироваться
        </a>
      </div>
```

В коде для регистрации имеется 4 колонки, для заполнения данных пользователя, колонка имени, электронной почты, пароля и подтверждение пароля. А также есть кнопка регистрации данных.

В пункте авторизации отображены 2 колонки, для заполнения зарегистрированных данных и кнопка авторизации. Код представлен в листинге 34.

Листинг 34– Код окна с пунктом авторизации

```
<div className="auth-form">
        <>
          <div className="mb-3 form-group">
            <i className="iconsax" data-icon="mail"></i>
            <label htmlFor="emailid" className="form-label">
             Логин
            </label>
            <input
              type="name"
```

```
            placeholder="Введите логин"
            className="form-control"
            id="emailid"
            onChange={(e) => handleFormFieldChange("name", e)}
          />
        </div>
        <div className="mb-2 form-group">
          <i className="iconsax" data-icon="lock-2"></i>
          <label htmlFor="password" className="form-label">
            Пароль
          </label>
          <input
            placeholder="Введите пароль"
            type="password"
            className="form-control"
            id="password"
            onChange={(e) => handleFormFieldChange("password", e)}
          />
        </div>
      </>
      {error && <div className="alert alert-danger">{error}</div>}
      <a
        data-cursor="pointer"
        className="btn-solid w-100 text-center mt-3"
        onClick={(e) => handleSignIn(e)}
      >
        Авторизоваться
      </a>
    </div>
  </div>
```

4.4. Реализация функционала веб-приложения

При вызове функции `Chat`, добавляется запись запроса для получения истории чата, после отправленного сообщения пользователем. Код функции представлен в листинге 35.

Листинг 35 – Функция `Chat`

```
if (token) {
      const fetchChatHistory = () => {
        axios({
          url: 'http://localhost:4000/chat/getUserThreads',
          method: 'GET',
          headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${token}`
          },
          withCredentials: true
        })
        .then(response => {
          setHistory(response.data);
          console.log(JSON.stringify(response.data));
        })
        .catch(error => {
          console.error('Error fetching chat history:', error);
        });
      };
```

```
        fetchChatHistory();

        const intervalId = setInterval(fetchChatHistory, 10000);

        return () => clearInterval(intervalId);
      } else {
          console.error('No access token found in cookies');
      }
  }, []);
```

Код функции проверяет, есть ли переменная token. Если token существует, то выполняется функция `fetchChatHistory`.

Функция `fetchChatHistory` отправляет `GET` запрос на URL `'http://localhost:4000/chat/getUserThreads'` с добавлением заголовка `Authorization`, в котором присутствует `Bearer` и значение `token`.

После получения ответа от сервера, данные сохраняются в переменную `history` и выводятся в консоль в формате JSON.

При возникновении ошибки при выполнении запроса, выводится сообщение об ошибке.

Код представленный в листинге 36. Использует синтаксис асинхронного программирования с использованием ключевого слова `await`, что обеспечивает ожидание ответа от сервера перед продолжением выполнения кода.

Листинг 36 – Код выполнения запроса

```
const response = await fetch("http://localhost:4000/chat", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      prompt: data.get("prompt"),
      threadId: null
    }),
    credentials: 'include' // This line ensures cookies are included in the
request

  });
```

Этот код выполняет HTTP-запрос методом `POST`. В заголовках запроса указывается тип содержимого.

В теле запроса передается JSON-объект, содержащий значение поля `prompt` из объекта данных и значение `threadId`. Также указан параметр, который гарантирует, что данные об аутентификации будут включены в запрос.

При вызове функции `handleDeleteOne`, производится удаление конкретной записи чата. Код функции представлен в листинге 37.

Листинг 37 – Функция `handleDeleteOne`

```
const handleDeleteOne = (e, id) => {
    e.preventDefault();

      axios({
        url: 'http://localhost:4000/chat/dropThread',
        method: 'DELETE',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${token}`
        },
        data: {
          "threadId": id
        },
        withCredentials: true
      })
      .then(response => {
        setHistory(response.data);
        console.log(JSON.stringify(response.data));
      })
      .catch(error => {
        console.error('Error fetching chat history:', error);
      });

    localStorage.removeItem("history");

      // window.location.reload();
  };
```

Этот код удаляет одну запись чата из истории чата с помощью API запроса `DELETE` и обновляет состояние истории чата после успешного удаления.

При вызове функции `handleDeleteAll`, происходит удаление всех записей чата. Код представлен в листинге 38.

Листинг 38 – Функция `handleDeleteAll`

```
const handleDeleteAll = () => {
    axios({
      url: 'http://localhost:4000/chat/dropAllThreads',
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`
      },
```

```
      withCredentials: true
    })
    .then(response => {
      setHistory(response.data);
      console.log(JSON.stringify(response.data));
    })
    .catch(error => {
      console.error('Error fetching chat history:', error);
    });
  localStorage.removeItem("history");
  };
```

При выполнении этой функции, используется библиотека axios для отправки запроса с определенными параметрами.

После успешного выполнения запроса, данные из ответа, сохраняются в состоянии `history`, а затем выводятся в консоль в формате JSON. В случае ошибки при выполнении запроса, выводится сообщение об ошибке в консоль.

После выполнения запроса, происходит удаление данных из `localStorage` с ключом `history`. Затем перезагружается страница веб-приложения.

При вызове функции `handleSignUp`,выполняется событие формы регистрации. Код функции представлен в листинге 39.

Листинг 39 – Функция `handleSignUp`

```
 const handleSignUp = async (e) => {
    if (Cookies.get('Bearer')) {
      setError("Вы уже авторизованы");
      window.location.href = "/chat";
      return;
  }
    e.preventDefault();
    if (user.password !== user.passwordConfirm) {
      console.log("Passwords do not match");
      return;
    }
    try {
      const response = await
axios.post("http://localhost:4000/user/register", {
      username: user.name,
      name: user.name,
      email: user.email,
      passwordHash: user.password
    });
    if (response.data.type === 'ERROR') {
      setError("Пользователь с таким именем уже существует");
    } else {
      setSuccess("Вы успешно зарегистрировались");
      setError("");
```

```
      console.log("Success:", response.data);
      const jsonObj = JSON.stringify(user);
      localStorage.setItem("userDetail", jsonObj);
      Cookies.set('Bearer', response.data.access_token, { expires: 7 }); //
7 дней
    }


    } catch (err) {
      setError("Registration failed");
      console.error(err);
    }
  };
```

Этот код позволяет обрабатывать регистрацию пользователя через отправку запроса на сервер и обработку ответа.

При вызове функции `handleSignIn`, выполняется обработчик для входа пользователя. Код функции представлен в листинге 40.

Листинг 40 – Функция `handleSignIn`

```
const handleSignIn = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post("http://localhost:4000/user/login",
{
        username: user.name,
        password: user.password,

      });
      if (response.data.type === 'ERROR') {
        setError("Неверный логин или пароль");
        setSuccess("");
      } else {
        setSuccess("Login successful!");
        setError("");
        console.log("Success:", response.data);
        const jsonObj = JSON.stringify(response.data.user);
        localStorage.setItem("userDetail", jsonObj);
        Cookies.set('Bearer', response.data.access_token, { expires: 7 });
// 7 дней
        window.location.href = "/";
      }
    } catch (err) {
      setError(("Неверный логин или пароль"));
      setSuccess("");
      console.error(err);
    }
```

Код функции отправляет `POST` запрос на сервер по указанному адресу с данными пользователя.

Получает ответ от сервера и проверяет его тип. Если тип данных устанавливает сообщение об ошибке и очищает сообщение об успехе.

В противном случае устанавливает сообщение об успехе, сохраняет данные пользователя хранилище и устанавливает `cookies` с токеном доступа на 7 дней и направляет пользователя на главную страницу.

При вызове функции `handleDropCounter`, выполняется обработчик для приобретения тарифа. Код функции представлен в листинге 41.

Листинг 41 – Функция `handleDropCounter`

```
const handleDropCounter = async () => {
    setIsLoading(true);
    if (!window.ethereum) {
      setIsLoading(false);
      return alert("Установите Metamask расширение");
    }

    try {
      await buyMembership(); // Вызываем функцию listMembership
      const response = await axios.put('user/dropCounter', {
        // Дополнительные данные, если необходимо
      });

      if (response.status === 200) {
        alert('Вы получили премиум тариф');
      } else {
        alert('Произошла ошибка при получении тарифа');
      }
    } catch (error) {
      alert('Произошла ошибка при получении тарифа');
    } finally {
      setIsLoading(false);
    }
  };
```

Функция `handleDropCounter` запускается при событии, например, клике на кнопку.

Сначала устанавливается состояние `isLoading` в значение `true`, чтобы указать, что процесс загрузки начался.

Затем происходит проверка наличия расширения Metamask. Если его нет, выводится сообщение об ошибке, и процесс завершается. Если расширение Metamask установлено, вызывается функция `buyMembership`.

После этого выполняется запрос к серверу с помощью метода `axios.put`. После получения ответа от сервера проверяется статус ответа. Если статус равен 200, выводится сообщение о получении премиум тарифа. В противном случае выводится сообщение об ошибке.

При вызове функции sendMessage, выполняется обработчик для отправки отзыва. Код функции представлен в листинге 42.

Листинг 42 – Функция sendMessage

```
const sendMessage = async () => {
    try {
      if (title.trim() === '' || message.trim() === '') {
        alert('Заголовок и текст отзыва не могут быть пустыми');
        return;
      }
      setIsSending(true);
      const str = localStorage.getItem('userDetail');
      const parsedObj = JSON.parse(str);
      const { data } = await axios.post('api/review/addReview', {
        title: title,
        text: message,
        opinion: true,
      });
      if (isLiked)
        await axios.put('api/review/addLike', {
          reviewId: data.id,
          userId: parsedObj.id,
        });
      if (isDisliked)
        await axios.put('api/review/addDislike', {
          reviewId: data.id,
          userId: parsedObj.id,
        });
        setIsSending(false);
        alert('Спасибо за ваш отзыв');

    } catch (error) {
      console.log(error);
      setIsSending(false);
      alert('Отзыв можно отправить только один раз');
    }
  };
```

Функция sendMessage является асинхронной, что означает, что она будет выполняться асинхронно и возвращает промис. Внутри функции используется блок try/catch, который отлавливает ошибки при выполнении кода внутри блока try.

Первым условием проверяется, пусты ли поля title и message. Если хотя бы одно из полей пустое, выводится сообщение об ошибке и функция прерывается с помощью return.

Далее устанавливается флаг isSending в значение true, чтобы отобразить пользователю, что отправка отзыва находится в процессе.
Затем из локального хранилища извлекается информация об авторизованном пользователе. С помощью запроса POST к API

отправляются данные для добавления нового отзыва с заголовком, текстом и установленным флагом `opinion` в значение `true`. Результат запроса записывается в переменную `data`.

После успешного выполнения запросов устанавливается флаг `isSending` в значение `false`, выводится сообщение об успешной отправке отзыва. В случае возникновения ошибки (например, пользователь пытается отправить отзыв несколько раз), выводится сообщение об ошибке и флаг `isSending` устанавливается в `false`.

Вывод по четвертой главе

В четвертой главе были описаны ключевые этапы создания системы. Для создания адаптивного и отзывчивого пользовательского интерфейса веб-сайта были использованы, язык программирования JavaScript и библиотека React. Для реализации функционала веб-приложения были использованы язык программирования TypeScript и фреймворк Nest.js. Для разработки базы данных, был использован PostgreSQL, взаимодействие с базой данных реализовано через TypeORM, и для обработки запросов была использована библиотека Axios. Также была использована технология блокчейн.

5. ТЕСТИРОВАНИЕ

5.1. Методы тестирования веб-приложения

Для тестирования веб-приложения использовался метод функционального тестирования.

Для тестирования была использована тестовая сеть Goerli.

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований.

В таблице 1 приведено тестирование интернет-приложения.

Таблица 1 – Тестирование веб-приложения

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Подключение кошелька без установленного в браузере расширения Metamask.	1. Зайти на веб-страницу при помощи вкладки «инкогнито», либо отключить/ расширение Metamask. 2. На веб-странице нажать на кнопку «Подключить кошелек».	На экран выведется сообщение с просьбой установить расширение Metamask.	Да
2	Подключение кошелька с установленным расширением Metamask, но без аутентификации в системе.	1. Зайти на веб-страницу со включенным расширением Metamask, без ввода пароля от аккаунта. 2. На веб-странице нажать на кнопку «Подключить кошелек».	На экран выведется сообщение о том, что авторизация в процессе, необходимо ввести данные.	Да
3	Регистрация нового пользователя с корректными данными.	1. Нажать на кнопку входа страницы регистрации. 2. Заполнить корректные данные. 3.Нажать на кнопку регистрации.	Пользователь успешно зарегистрирован в систему.	Да

Продолжение таблицы 1

4	Авторизация зарегистрированного пользователя с правильными учетными данными.	1. Зайти на форму авторизации пользователя. 2. Ввести корректные данные. 3. Нажать на кнопку авторизации.	Пользователь успешно вошел в личный аккаунт.	Да
5	Попытка авторизации с неверным паролем.	1. Ввести неверные данные. 2. Нажать на кнопку авторизации.	Выведено сообщение об ошибке с пояснением.	Да
6	Выход авторизованного пользователя из аккаунта.	Нажать на кнопку выхода после использования нейронной модели.	Пользователь успешно вышел из личного аккаунта.	Да
7	Попытка приобрести подписку, для использования моделью, до подключения кошелька.	1. Не нажимать кнопку «Подключить кошелек». 2. Кликнуть на надпись «Выбрать».	На экран выведется сообщение «Error» с ссылкой на расширение Metamask.	Да
8	Приобретение подписки.	1. Подключить кошелек. 2. Нажать на кнопку «Выбрать». 3. В появившемся окне нажать кнопку «Confirm».	Расширение Metamask выведет на экран детали сделки и при нажатии на «Confirm», на кнопке появится сообщение success и браузер уведомит об успешной операции., для приобретения тарифа.	Да
9	Просмотр истории ответов.	1. Зайти на страницу нейронной модели. 2. Выбрать раздел истории.	Откроется раздел с историей ответов нейронной модели.	Да
10	Отправка запроса с текстом.	1. На странице чат-бота найти форму текстового ввода. 2. Написать в форму текстовое сообщение. 3. После ввода текста, отправить тсообщение.	После обработки запроса, нейронная модель отправит ответ на запрос текстового сообщения.	Да

Окончание таблицы 1

11	Выбор записи конкретного запроса.	1. После отправки текстового сообщения, найти запись запроса. 2. Нажать на необходимую запись запроса.	После клика на запись, откроется окно только с выбранным запросом.	Да
12	Удаление записи конкретного запроса.	1. После открытия записи запроса, найти кнопку удаления возле записи. 2. Нажать на кнопку удаления.	После нажатия кнопки, запись запроса удалится из истории.	Да
13	Удаление истории запросов.	1. После отправки запросов, найти кнопку удаления всей истории. 2. Нажать на кнопку очищение истории запросов.	После нажатия кнопки, все записи запросов, отправленные пользователем удалятся.	Да
14	Написать отзыв.	1. После открытия модального окна, необходимо написать тему и текст отзыва. 2. Нажать на кнопку «Отправить отзыв».	После нажатия кнопки, отзыв пользователя отправится с выводом сообщения на экран.	Да
15	Изменить данные.	1. После открытия раздела настроек, необходимо заполнить поля для изменения учетных данных. 2. Нажать на кнопку «Обновить».	После нажатия кнопки, учетные данные изменятся с выводом сообщения на экран.	Да

Вывод по пятой главе

В пятой главе было проведено функциональное тестирование веб-приложения. Все тесты на функционал приложения пройдены без ошибок, и были получены корректные результаты.

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы было разработано веб-приложение на основе технологии блокчейн, которое объединяет различные функции и обеспечивает удобство использования для пользователя. При этом были выполнены следующие задачи.

1. Произведен обзор существующих приложений по предметной области.
2. Проведен анализ предметной области, что позволило определить требования и функциональность приложения.
3. Выполнена верстка страниц, обеспечивающая правильное отображение и взаимодействие элементов интерфейса.
4. Реализованы функциональные возможности веб-приложения.
5. Проведено тестирование разработанного веб-приложения на тестовой сети Metamask.

В результате была получена система, которая обеспечивает пользователю задавать запросы нейронной модели и возможность с ними взаимодействовать.

Система соответствует заявленному функционалу и обладает потенциалом для дальнейшего его расширения.

ЛИТЕРАТУРА

1. Прасти Н. Блокчейн. Разработка приложений. // БХВ-Петербург, 2018. – 252 с.

2. Равал С. Децентрализованные приложения. Технология Blockchain в действии. // Питер, 2017. – 240 с.

3. Шлемин А. NFT. Технология, которая изменит мир. // Издательские решения, 2021. – 170 с.

4. Фролов А. В. Создание смарт-контрактов Solidity для блокчейна Etherium. Практическое руководство. // ЛитРес, 2019. – 240 с.

5. Сонг Д. Python для программирования криптовалют. // Диалектика, 2020. – 368 с.

6. Бурков А. Ethereum: работа с сетью, смарт-контракты и распределенные приложения. // ООО «Лаборатория цифровой трансформации», 2020. – 420 с.

7. Андреас А. Осваиваем Ethereum. Создание смарт-контрактов и децентрализованных приложений. // Бомбора, 2021. – 512 с.

8. Metamask. [Электронный ресурс] URL: https://metamask.io/ (дата обращения: 05.02.2024 г.).

9. YandexGPT. [Электронный ресурс] URL: https://cloud.yandex.ru/ (дата обращения: 07.02.2024 г.).

10. StateOfDApps. [Электронный ресурс] URL: https://www.stateofthedapps.com/ (дата обращения: 08.02.2024 г.).

11. Руководство по web3.js. [Электронный ресурс] URL: https://web3js.readthedocs.io/(дата обращения: 09.02.2024 г.).

12. Документация React [Электронный ресурс] URL: https://reactjs.org/ (дата обращения: 10.02.2024 г.).

13. Документация CSS-in-JS библиотеки styled-components. [Электронный ресурс] URL: styled-components.com (дата обращения: 14.02.2024 г.).

14. Документация Sass. [Электронный ресурс] URL: https://sass-lang.com/ (дата обращения: 17.02.2024 г.).

15. Документация линтера ESLint. [Электронный ресурс] URL: https://eslint.org/ (дата обращения: 26.02.2024 г.).

16. Пакетный менеджер npm.js. [Электронный ресурс] URL: https://www.npmjs.com/ (дата обращения: 26.02.2024 г.).

17. Документация сборщика ESBuild. [Электронный ресурс] URL: https://esbuild.github.io/ (дата обращения: 26.02.2024 г.).

18. Flanagan D. JavaScript: The Definitive Guide, 7th Edition. // O'Really Media Inc 2020. – 1032 с.

19. Antonopoulos A. Mastering Bitcoin. // O'Really Media, 2014. – 298 с.

20. Пакет html-to-image. [Электронный ресурс] URL: https://www.npmjs.com/package/html-to-image (дата обращения: 26.02.2024 г.).

21. Сервис для размещения веб-страниц GitHub Pages. [Электронный ресурс] URL: https://pages.github.com/ (дата обращения: 02.04.2024 г.).

22. Документация фреймворка NextJs. [Электронный ресурс] URL: https://nextjs.org/ (дата обращения: 09.04.2024 г.).

23. Документация PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org/docs/ (дата обращения: 18.04.2024 г.).

24. Документация фреймворка NestJs. [Электронный ресурс] URL: https://nestjs.com/ (дата обращения: 22.04.2024 г.).

25. Документация TypeScript. [Электронный ресурс] URL: https://www.typescriptlang.org/ (дата обращения: 23.04.2024 г.).

26. Документация Axios. [Электронный ресурс] URL: https://axios-http.com/ (дата обращения: 23.04.2024 г.).

27. Документация TypeORM. [Электронный ресурс] URL: https://typeorm.io/ (дата обращения: 27.04.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования для веб-приложения, основанного на технологии блокчейн приведена в таблицах 1–10.

Таблица 1 – Спецификация прецедента «Регистрация аккаунта»

Альтернативный поток: Регистрация аккаунта.
ID: 1
Краткое описание: Регистрация аккаунта пользователя.
Главные актеры: Неавторизованный пользователь.
Предусловия: Неавторизованный пользователь подключил крипто-кошелек и перешел на страницу регистрации.
Основной поток: 1. Прецедент начинается, когда пользователь нажимает на кнопку «Начать пользоваться» и выбирает окно с регистрацией аккаунта. 2. Если у неавторизованного пользователя не получилось зарегистрировать данные, то вызывается ошибка с сообщением, что пользователь уже существует.
Постусловия: Неавторизованный пользователь зарегистрировал аккаунт.

Таблица 2 – Спецификация прецедента «Авторизация аккаунта»

Альтернативный поток: Авторизация аккаунта.
ID: 2
Краткое описание: Авторизация аккаунта пользователя.
Главные актеры: Неавторизованный пользователь.
Предусловия: Неавторизованный пользователь подключил крипто-кошелек и перешел на страницу авторизации.
Основной поток: 1. Прецедент начинается, когда пользователь нажимает на кнопку «Начать пользоваться» и выбирает окно с авторизацией аккаунта. 2. Неавторизованный пользователь вводит данные при регистрации, и нажимает на кнопку «Авторизоваться».
Постусловия: Неавторизованный пользователь выполнил авторизацию аккаунта.

Таблица 3 – Спецификация прецедента «Подключить крипто кошелек»

Прецедент: Подключить крипто кошелек.
ID: 3

Краткое описание: Подключить крипто-кошелек к расширению Metamask.
Главные актеры: Неавторизованный пользователь.
Предусловия: Неавторизованный пользователь нажал на кнопку «Подключить крипто кошелек».
Основной поток: 1. Прецедент начинается после нажатия кнопки «Подключить крипто кошельк». 2. Система сообщает неавторизованному пользователю, что нужно войти в аккаунт крипто-кошелька.
Постусловия: Система вывела сообщение.

Таблица 4 – Спецификация прецедента «Приобрести тариф»

Прецедент: Приобрести тариф.
ID: 4
Краткое описание: Оплата через крипто кошелек для доступа к нейронной модели.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь подключил крипто кошелек.
Основной поток: 1. Прецедент начинается, когда пользователь нажимает на кнопку «Выбрать тариф». 2. Если авторизованный пользователь подключил крипто кошелек, то появляется окно в расширении Metamask, для оплаты смарт-контракта, чтобы приобрести тариф. 3. После успешной оплаты, авторизованный пользователь переходит на ссылку с нейронной моделью.
Постусловия: Неавторизованный пользователь получил доступ к нейронной модели.

Таблица 5 – Спецификация прецедента «Выход»

Прецедент: Выход.
ID: 5
Краткое описание: Выход из аккаунта авторизованного пользователя на главную страницу.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь нажал на кнопку выхода.

Основной поток: 1. Прецедент начинается, когда авторизованный пользователь зашел на страницу с нейронной моделью. 2.После перехода на страницу с нейронной моделью, авторизованный пользователь нажимает на кнопку выхода. 3. После нажатия кнопки, авторизованный пользователь выходит из аккаунта и переходит на главную страницу приложения.
Постусловия: Неавторизованный пользователь получил доступ к нейронной модели.

Таблица 6 – Спецификация прецедента «Отправить запрос с текстом»

Прецедент: Отправить запрос с текстом.
ID: 6
Краткое описание: отправка запроса с текстом в нейронной модели.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь ввел запрос с текстом и нажал на кнопку отправки.
Альтернативный поток: 1. Прецедент начинается после перехода на страницу с нейронной моделью. 2. Авторизованный пользователь ввел текст, для отправки запроса. 3. Авторизованный пользователь нажал на кнопку отправки запроса с введенным текстом.
Постусловия: Авторизованный пользователь получил ответ от нейронной модели, после отправки запроса с текстом.

Таблица 7 – Спецификация альтернативного потока «Выбрать запрос»

Альтернативный поток: Выбрать запрос.
ID: 7
Краткое описание: Выбрать объект с запросом, отправленный пользователем.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь отправил запрос с текстом.
Основной поток: 1. Прецедент начинается, когда пользователь отправил запрос с введенным текстом сообщения. 2. После отправки запроса, появляется запись с отправленным запросом текста сообщения.
Постусловия: Авторизованный пользователь выбрал запись с запросом отправленного текста сообщения.

Таблица 8 – Спецификация альтернативного потока «Удалить запрос»

Альтернативный поток: Удалить запрос.
ID: 8
Краткое описание: Удаление объекта запроса, отправленного текста сообщения.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь нажал на кнопку удаления запроса.
Альтернативный поток: 1. Альтернативный поток начинается, после добавление записи с отправленным запросом текста сообщения. 2. Авторизованный пользователь нажал на кнопку удаления.
Постусловия: Система удалила запись с запросом отправленного текста сообщения.

Таблица 9 – Спецификация альтернативного потока «Очистить запросы»

Альтернативный поток: Очистить запросы.
ID: 9
Краткое описание: Очищение объектов запроса, отправленного текста сообщения.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь нажал на кнопку очищения запросов.
Альтернативный поток: 1. Альтернативный поток начинается, после добавление записей с отправленным запросом текста сообщения. 2.Авторизованный пользователь нажал на кнопку очищения запросов.
Постусловия: Система очистила запись с запросом отправленного текста сообщения.

Таблица 10 – Спецификация альтернативного потока «Просмотр истории»

Альтернативный поток: Просмотр истории.
ID: 10
Краткое описание: Просмотр истории ответов, отправленные нейронной моделью.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь нажал на иконку раздела истории.
Альтернативный поток: 1. Альтернативный поток начинается, когда пользователь заходит в раздел истории после отправки запроса.
Постусловия: Авторизованный пользователь перешел в раздел с историей ответов.

Таблица 11 – Спецификация альтернативного потока «Написать отзыв»

Альтернативный поток: Написать отзыв.
ID: 11
Краткое описание: Написать отзыв для оценки приложения.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь нажал на иконку раздела отзывы.
Альтернативный поток: 1. Альтернативный поток начинается, когда пользователь заходит в раздел отзыва. 2. Заполняет поля для отзыва. 3. Нажимает на кнопку «Отправить».
Постусловия: Авторизованный пользователь отправил отзыв для оценки приложения.

Таблица 12 – Спецификация альтернативного потока «Изменить данные»

Альтернативный поток: Изменить данные.
ID: 12
Краткое описание: Изменение учетных данных пользователя.
Главные актеры: Авторизованный пользователь.
Предусловия: Авторизованный пользователь нажал на иконку раздела настройки.
Альтернативный поток: 1. Альтернативный поток начинается, когда пользователь заходит в раздел настроек. 2. Заполняет поля для редактирования данных. 3. Нажимает на кнопку «Обновить».
Постусловия: Авторизованный пользователь изменил учетные данные.

Приложение Б. Скриншоты веб-приложения

Скриншоты с оформлением веб-приложения представлены на рисунках 1–12.

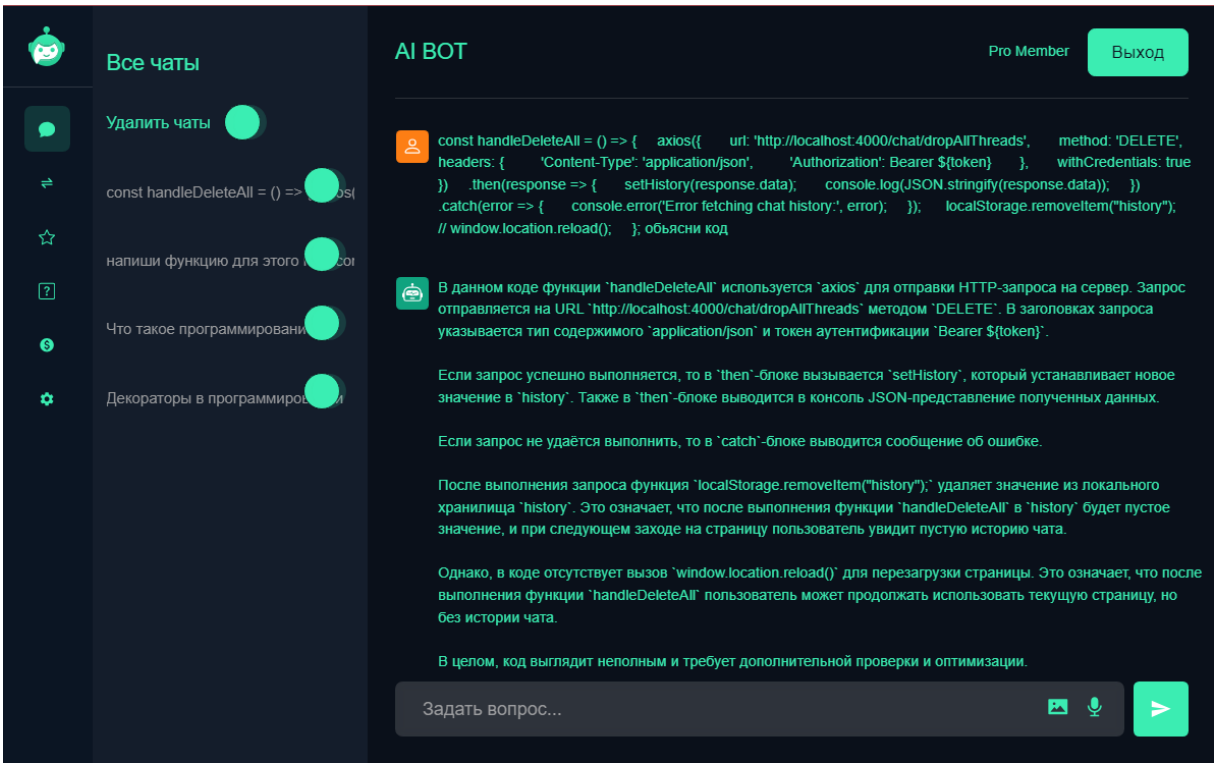

Рисунок 1 – Главный чат

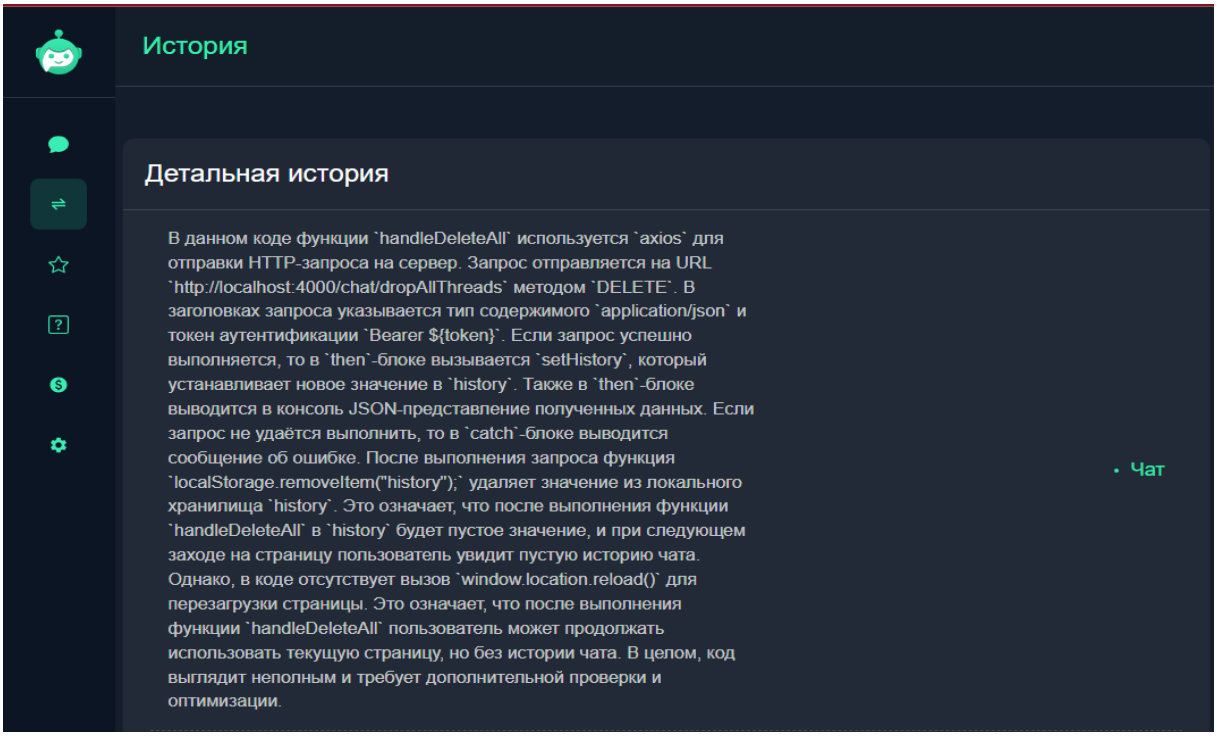

Рисунок 2 – История чата

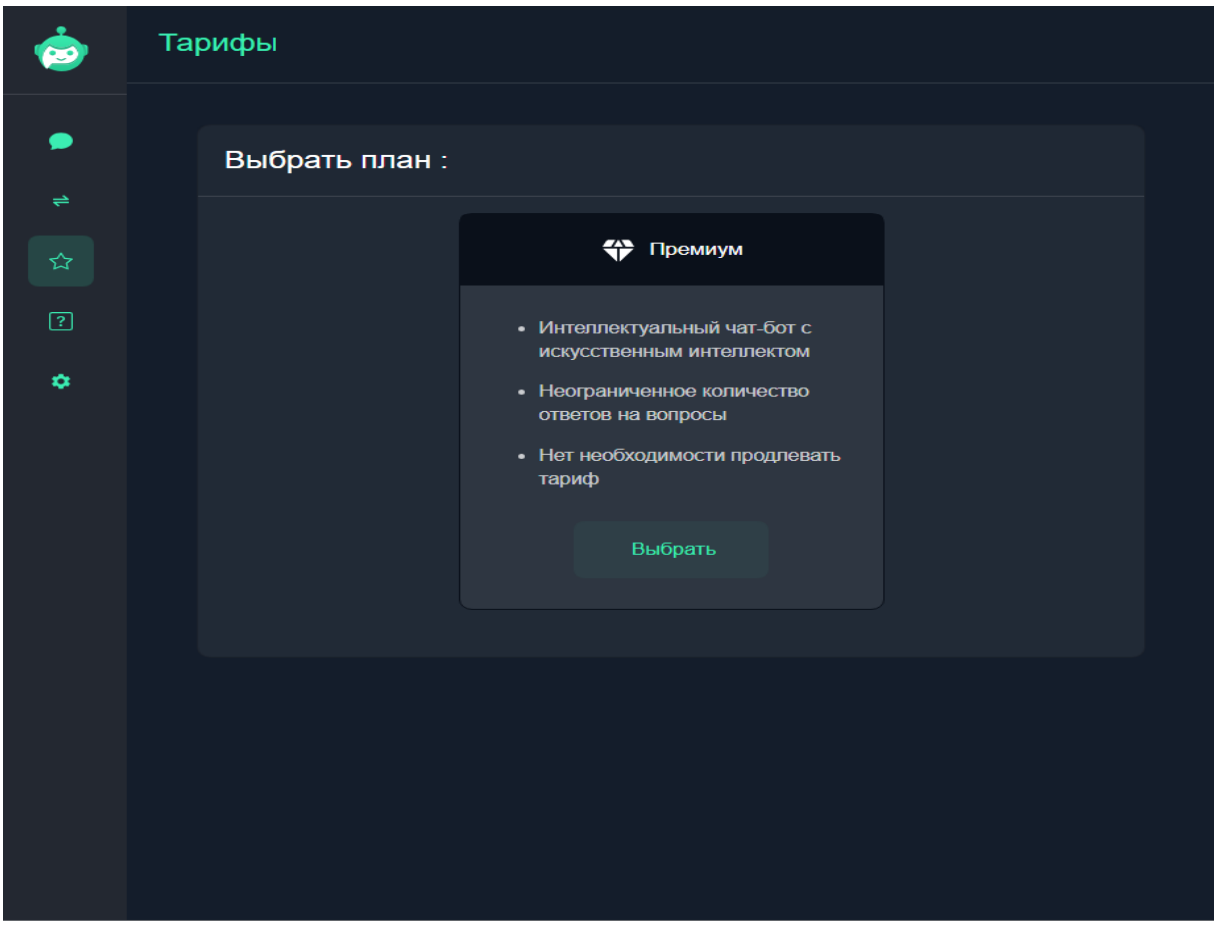

Рисунок 3 – Форма тарифа

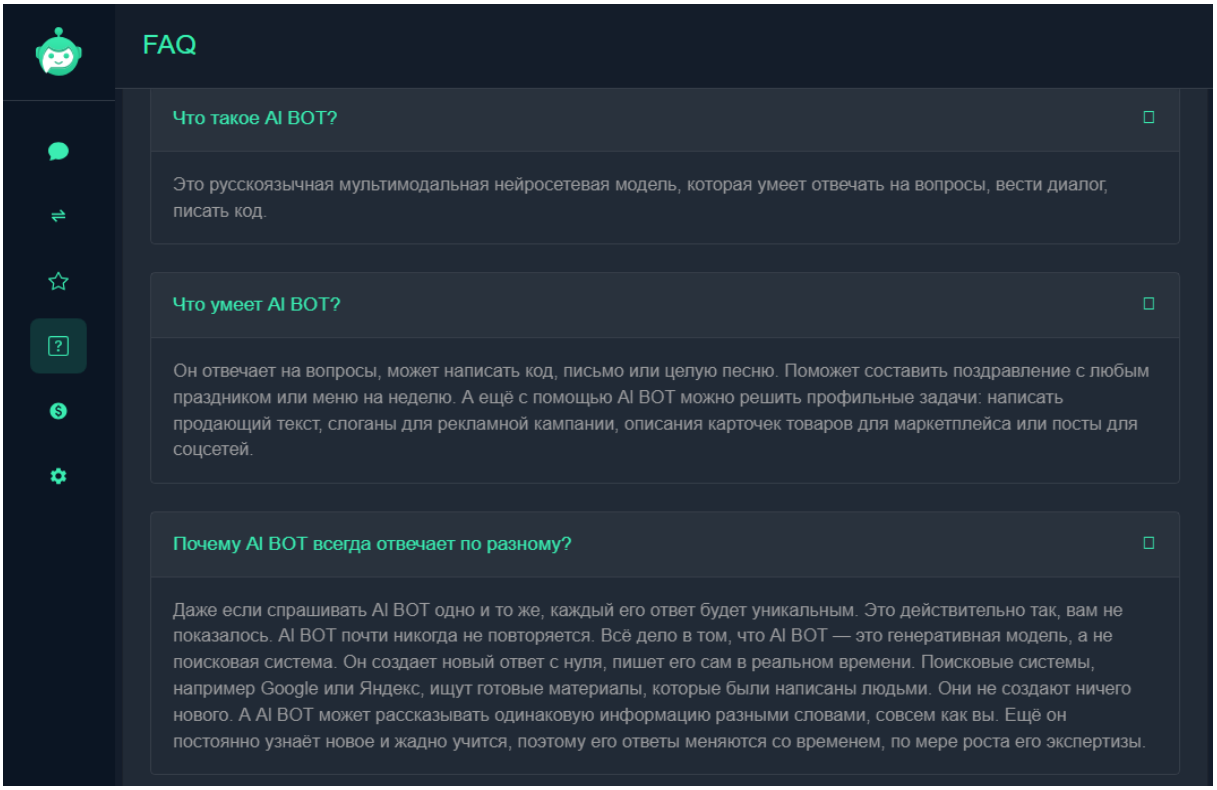

Рисунок 4 – Общие вопросы

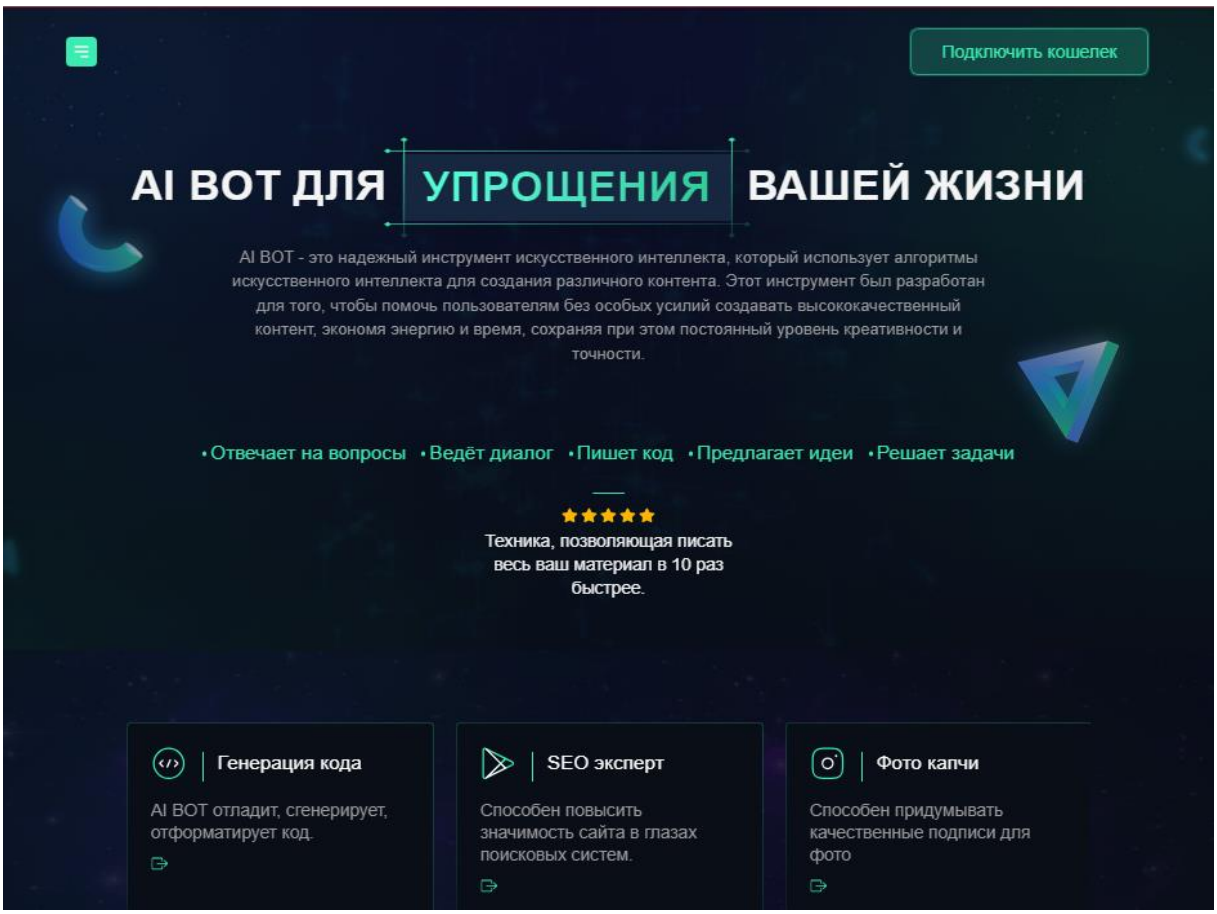

Рисунок 5 – Главная страница

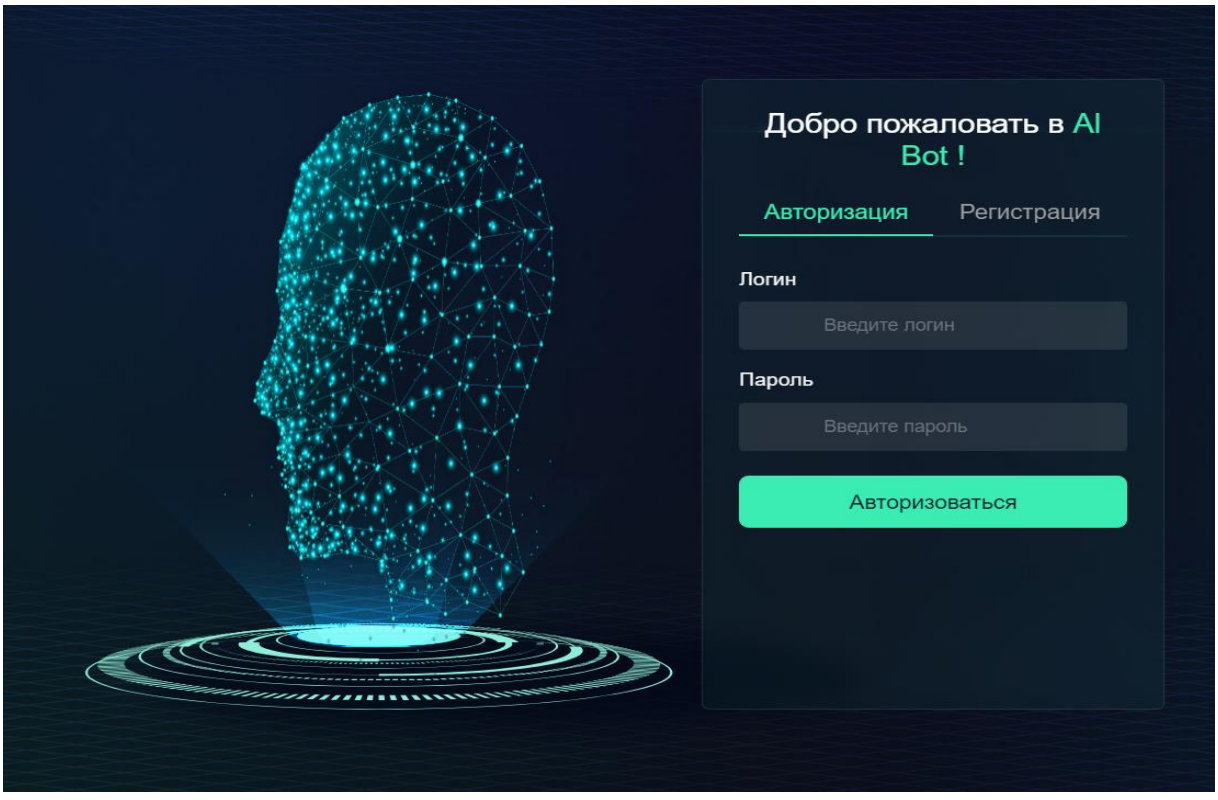

Рисунок 6 – Страница авторизации и регистрации

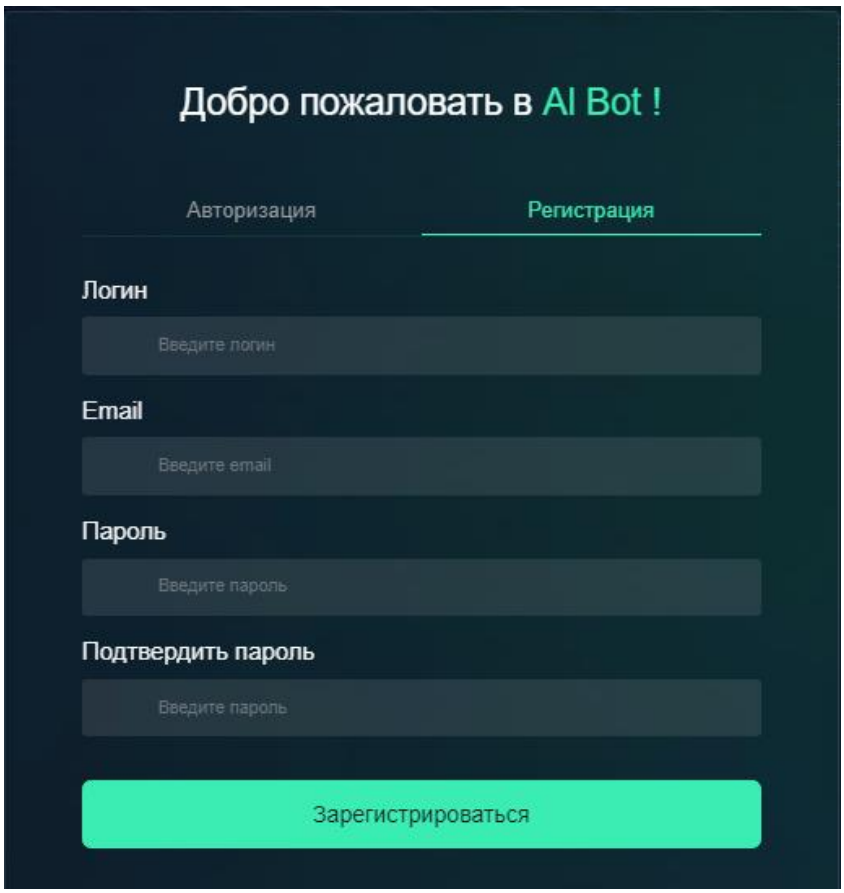

Рисунок 7 – Форма регистрации

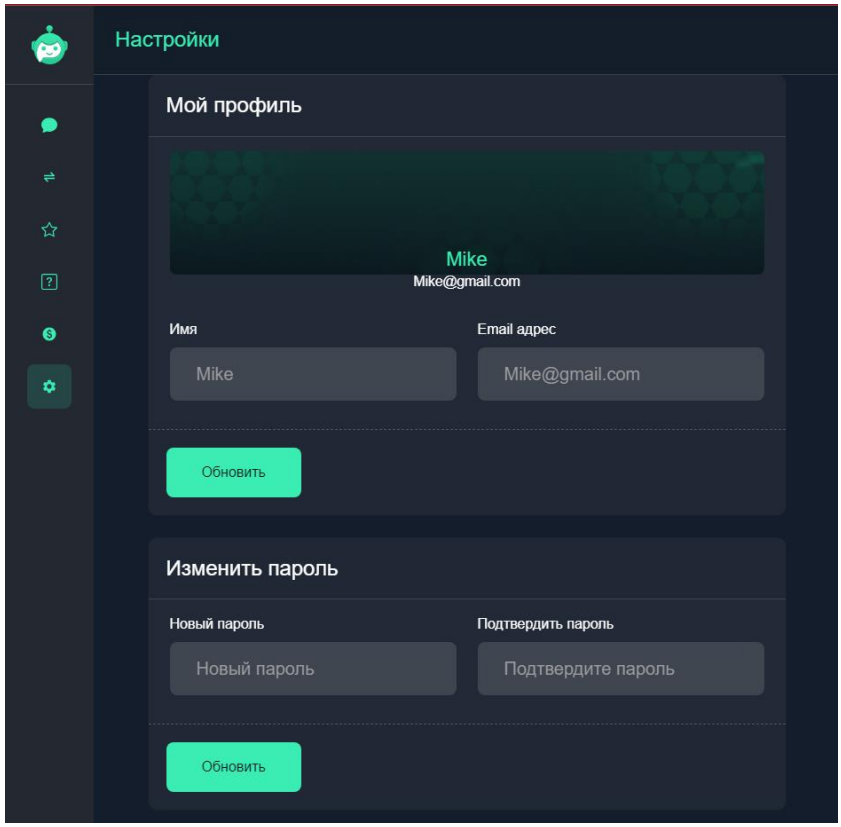

Рисунок 8 – Настройки пользователя

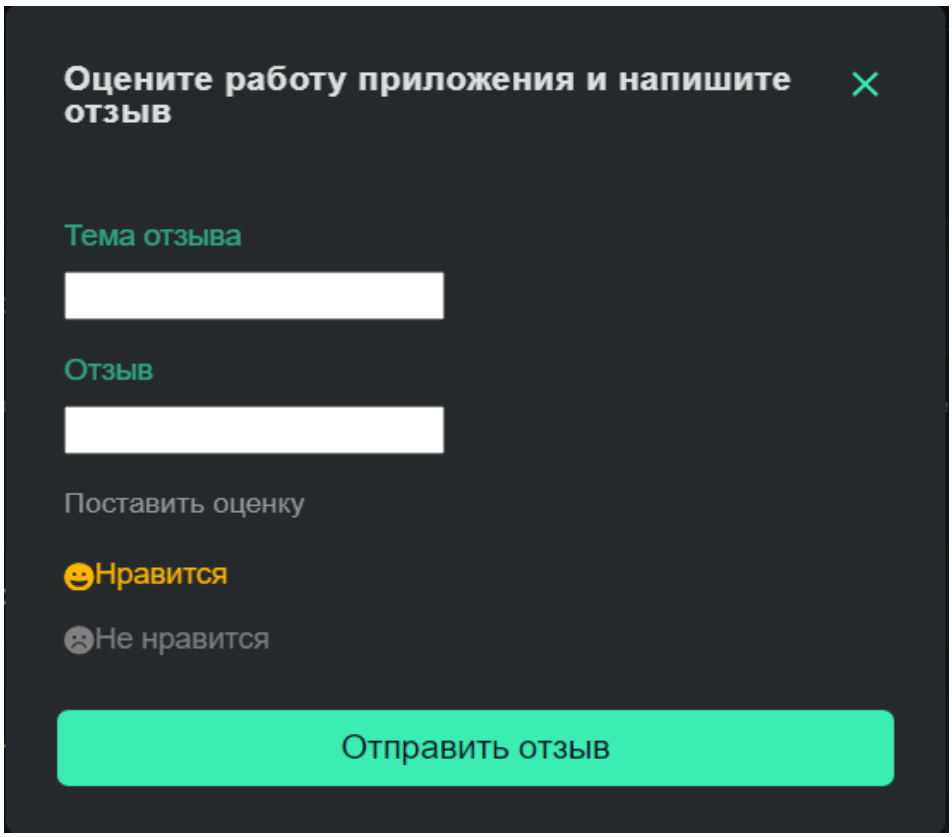

Рисунок 9 – Окно отзыва

Рисунок 10 – Меню чата для мобильных версий

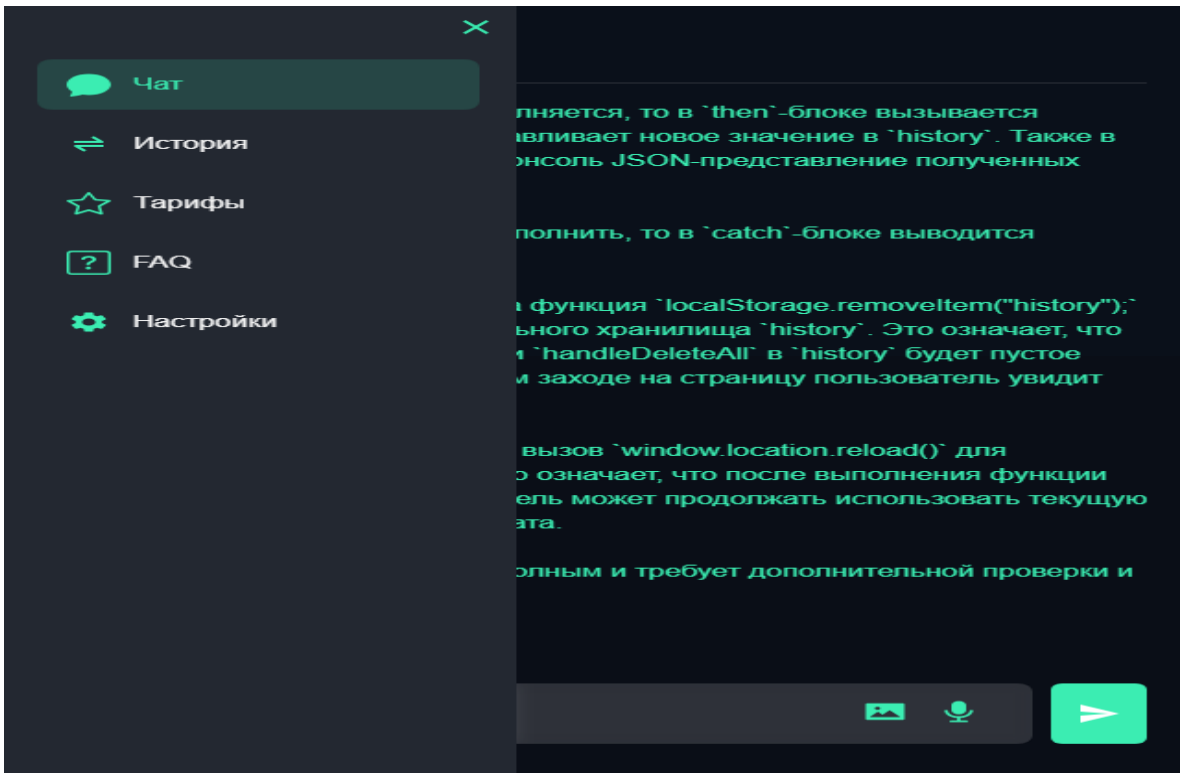

Рисунок 11 – Главное меню для мобильных версий

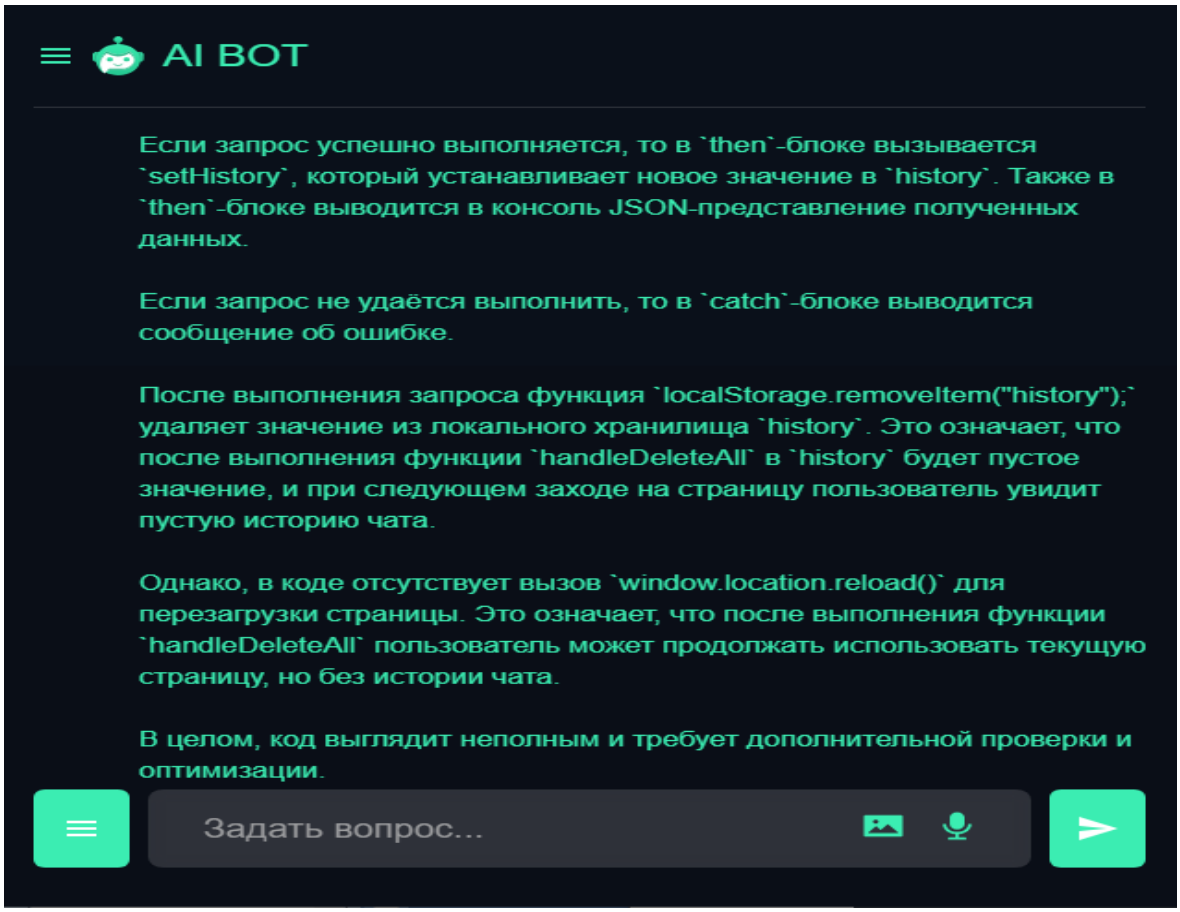

Рисунок 12 – Главный чат для мобильных версий