Министерство образования Новосибирской области
ГБПОУ НСО «Новосибирский авиационный технический колледж имени Б.С. Галущака»

РЕАЛИЗАЦИЯ ПОДПИСОЧНОЙ МОДЕЛИ В ЧАТ-ПРИЛОЖЕНИИ НА ОСНОВЕ ФРЕЙМВОРКА CHAINLIT

Отчёт по производственной практике
по ПМ.01 Разработка модулей программного обеспечения
для компьютерных систем

НАТКиГ.720800.010.000

Разработал:
студент группы ПР-23.106
Кретинин Димтрий Александрович

2025

Содержание

Введение

В современном мире, с высокой тенденцией развития искусственного интеллекта на просторах интернета активно начали появляться чат-боты именуемые, как – «ChatGPT». Данный чат-бот предоставляет возможность быстро получить ответ практически на любой вопрос. В соответствии с развитием этой технологии постепенно в среду программистов начало вводиться такое понятие как «VibeCoding». Данный процесс является программированием с использованием нейросетей для быстрого написания какой-либо программы.

Компания ООО «ДЕФФАН» занимается разработкой собственного бота на основе OpenAI, задумка которого заключается в быстрой и эффективной разработке проектов с использованием языка Python, помощи в развёртывании и дальнейшего поддержания различных проектов.

Целью проекта является создание подписочной модели, которая впоследствии будет регулировать возможности пользователя в общении с чат-ботом. Для реализации этой цели выявлены следующие задачи:

− проанализировать документацию API предоставленного платёжного шлюза (Tochka);

− изучение фреймворка Chainlit, который позволяет быстро и качественно разработать графический интерфейс и серверную составляющую бота.

− доработать схему БД для качественного и удобного хранения данных о подписках пользователей.

Чтобы успешно справиться с разработкой, необходимо обладать следующими навыками:

− опыт работы с такими плагинами как: sqlalchemy, alembic, asyncpg, pydantic

− опыт работы с базами данных SQL;

− средний уровень разработки на языке программирования Python с опытом разработки ботов;

− понимание работы HTTP запросов и умение работать с открытыми API;

− умение работать с системой контроля версий Git.

Для более просто восприятия терминологии, следует прочитать нижеприведённые термины:

– Data layer – некая прослойка во фреймворке, предоставляющая возможность сохранения данных.

Для выполнения данной задачи следует знать следующие понятия:

− переменные окружения – общий файл конфигурации в проекте, обычно хранящий в себе различные пароли к БД или же ключи для сторонних API;

− API – программный интерфейс приложения, являющийся прослойкой между приложением и сервером.

1 Описание предметной области

В настоящее время очень стремительно начала развиваться сфера искусственного интеллекта, довольно большое количество компаний вводят в свою инфраструктуру приложения, где присутствует бот с встроенным ИИ. Будь то интеграция с OpenAI с использованием их API. Или же собственные разработки в данной сфере.

Также с этой тенденцией начало часто использоваться понятие «VibeCoding», обозначающее, что человек, будучи, не являясь программистом разрабатывает какое-либо приложение при помощи ChatGPT.

Компания «ООО Деффан» также решила предоставить свой продукт для так называемых Вайбкодеров, который будет предлагать пользователем простое и быстрое написание кода на языке программирования Python, развертывание и дальнейшая поддержка проекта, разбор GitHub репозитория. Некоторые из этих функций будут доступны пользователю только по подписке.

Продукт разрабатывается на Python, за графическую основу взят фреймворк Chainlit, автоматически генерирующий графическую составляющую проекта и предоставляющий несколько схем БД и несколько способов сохранения данных. Изначальный весь проект уже имеет файл Docker-compose, позволяющий собрать всё решение при помощи одной команды. В данном случае генерируется 4 контейнера. Первый содержит в себе базу данных на PostgreSQL, второй само приложение, третий является вспомогательным и производит только миграцию с помощью инструмента под названием Prisma. Последний является скриптом для системы Linux.

2 Проектирование информационной системы

2.1 Постановка задачи

Целью данного проекта является полная разработка подписочной модели в уже существующем продукте. Для её реализации нужно выполнить следующие задачи:

– доработка схемы БД, для фиксации подписок с соблюдением 3НФ;

– настройка оплаты при помощи формирования платёжных ссылок используя платёжный шлюз Tochka;

– смена способа хранения данных с базового, встроенного в фреймворк Chainlit, на систему, которая использует SQLAlchemy.

2.2 Диаграмма прецедентов и спецификация

На рисунке 1 представлена диаграмма прецедентов, регламентирующая функциональные возможности пользователей системы. Диаграмма прецедентов является одним из видов представления модели поведения системы, поскольку здесь отражены варианты использования и взаимодействие актора с собственными прецедентами

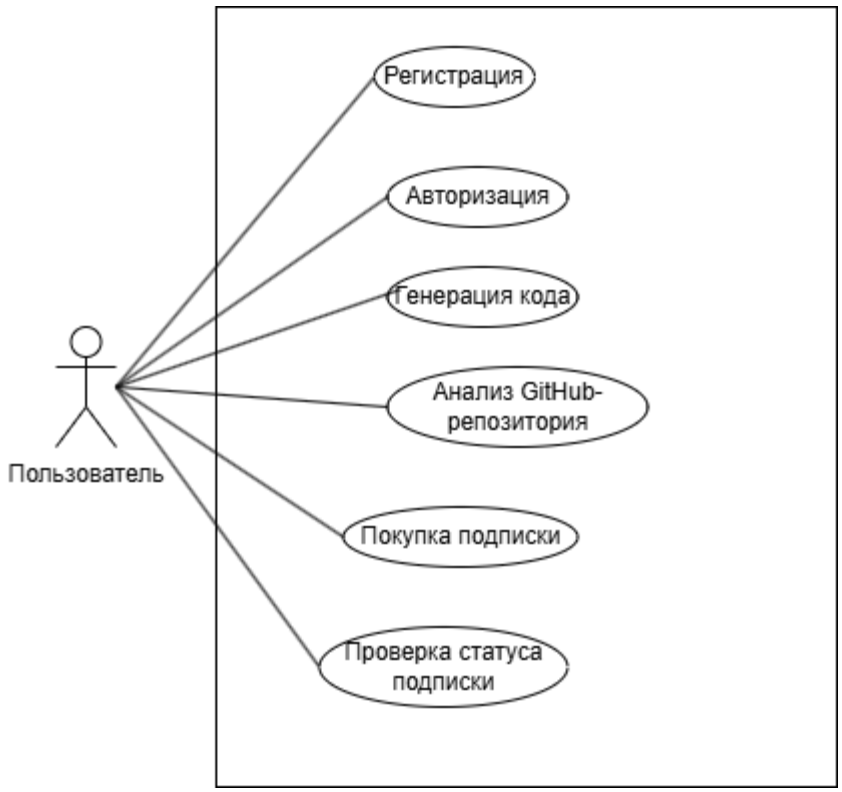

Рисунок 1–Диаграмма прецедентов

Для каждого прецедента созданы спецификации. Спецификации прецедентов позволяют детализировать прецедент, указав, что должна делать система, когда актор инициировал прецедент, а также возможные варианты поведения в исключительных ситуациях. Спецификации прецедентов представлены в таблицах

Таблица 1 – Спецификация прецедента «Регистрация»

Раздел	Описание
Краткое описание	Пользователь создает учетную запись в системе
Действующие лица	Пользователь
Предусловие	Пользователь не авторизован в системе
Основной поток	1. Пользователь вводит email и пароль 2. Система проверяет уникальность email 3. Учетная запись создается, данные сохраняются в базу данных
Альтернативный поток	Если email уже занят, система предлагает восстановить пароль или использовать другой email
Постусловие	Пользователь зарегистрирован и может авторизоваться

Таблица 2 – Спецификация прецедента «Генерация кода»

Раздел	Описание
Краткое описание	Пользователь запрашивает генерацию кода через чат-интерфейс
Действующие лица	Пользователь
Предусловие	Пользователь авторизован и имеет доступ к функционалу (по подписке/бесплатно)
Основной поток	1. Пользователь вводит текстовое описание задачи; 2. Система генерирует код и возвращает его пользователю; 3. Код сохраняется в истории запросов.
Альтернативный поток	Если запрос некорректен, ИИ уточняет детали у пользователя
Постусловие	Пользователь получает рабочий код

Таблица 3 – Спецификация прецедента «Анализ GitHub-репозитория»

Раздел	Описание
Краткое описание	Пользователь запрашивает анализ своего репозитория на GitHub
Действующие лица	Пользователь
Предусловие	Пользователь авторизован и имеет свой GitHub-аккаунт
Основной поток	1. Пользователь вводит ссылку на репозиторий; 2. Система получает данные о репозитории; 3. Система анализирует код и формирует отчет.
Альтернативный поток	Если репозиторий приватный, система запрашивает доступ
Постусловие	Пользователь получает отчет с рекомендациями по улучшению

Таблица 4 – Спецификация прецедента «Покупка подписки»

Раздел	Описание
Краткое описание	Пользователь оформляет платную подписку на премиум-функции
Действующие лица	Пользователь
Предусловие	Пользователь авторизован и имеет активную платежную карту
Основной поток	1. Пользователь выбирает тарифный план; 2. Система перенаправляет на страницу оплаты; 3. После успешной оплаты доступ к функциям расширяется.
Альтернативный поток	Если оплата не прошла, система предлагает повторить попытку или выбрать другой способ оплаты
Постусловие	Подписка активирована, премиум-функции доступны

Таблица 5 – Спецификация прецедента «Проверка статуса подписки»

Раздел	Описание
1	2
Краткое описание	Пользователь проверяет активность и срок действия подписки
Действующие лица	Пользователь

Продолжение таблицы 5

1	2
Предусловие	Пользователь авторизован
Основной поток	1. Пользоватеель открывает раздел проверки статуса подписки; 2. Система отображает текущий статус подписки.
Альтернативный поток	Если подписка отсутствует, система предлагает ознакомиться с тарифами
Постусловие	Пользователь информирован о статусе подписки

3 Разработка приложения

3.1 Проектирование базы данных

Фреймворк Chainlit предлагает следующую схему БД, которая обеспечивает полное сохранение данных. Ниже, на рисунке представлена ERD диаграмма данной базы.

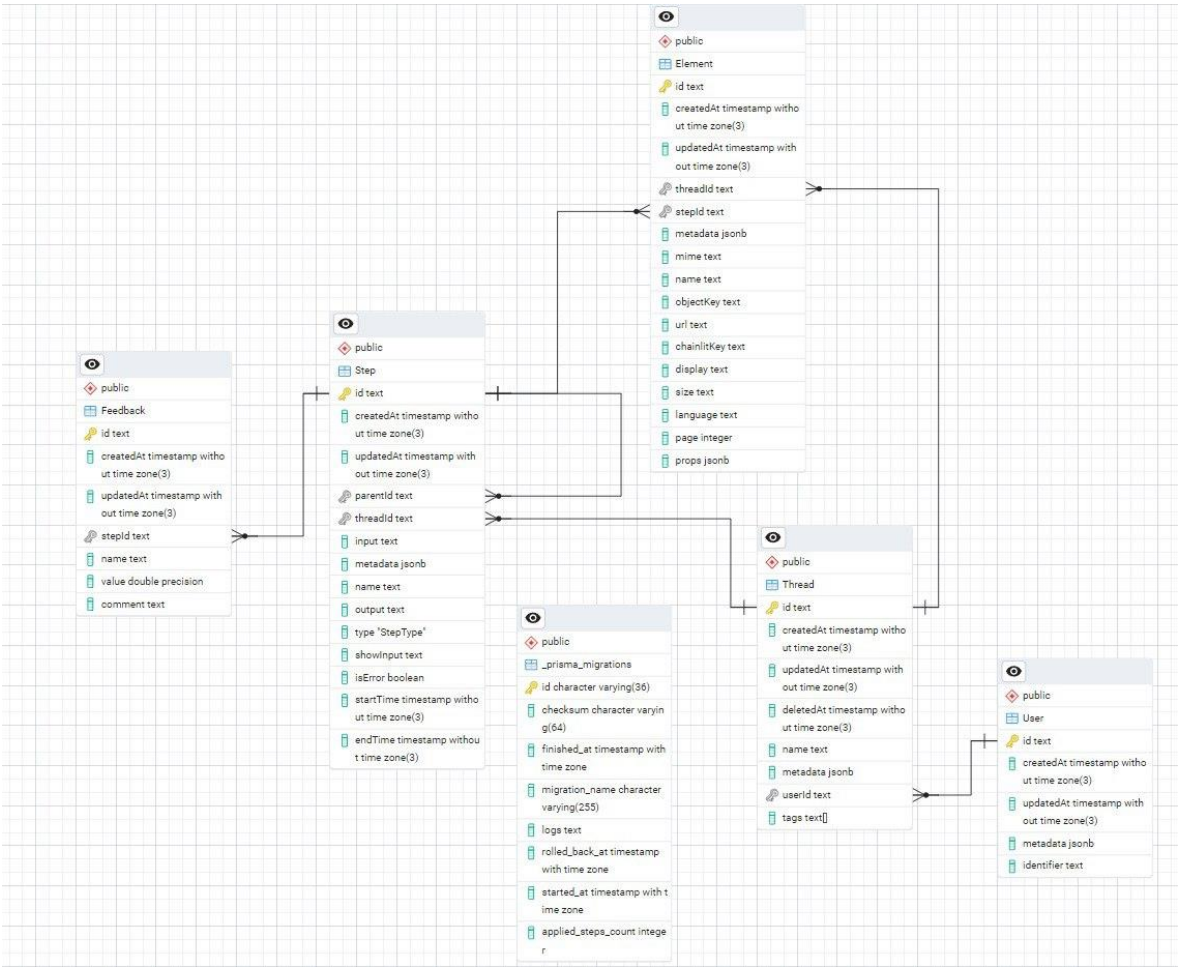

Рисунок 2 – изначальная схема БД

Данная схема позволяет сохранять все нужные данные об истории чатов и авторизованных пользователях.

Заказчик предоставил задание совершить переход с официального Data Layer на созданный пользователями слой с использованием SQLAlchemy.

Для осуществления перехода пришлось полностью поменять существующую схему на предоставленную для корректной работы слоя с SQLAlchemy.

Также, для правильного хранения подписок принято решение ввести 3 новые сущности, такие как:

– SubTypes, таблица, хранящая в себе типы подписок по их продолжительности;

– Payments хранит в себе данные об оплате, а точнее, сумму и идентификатор операции;

– Subscription объединяет в себе эти таблицы и содержит данные о начале, окончании подписки, id операции, id типа подписки, идентификатор пользователя, а также, будет ли возобновляться подписка автоматически или нет.

На рисунке ниже, представлена обновлённая схема БД.

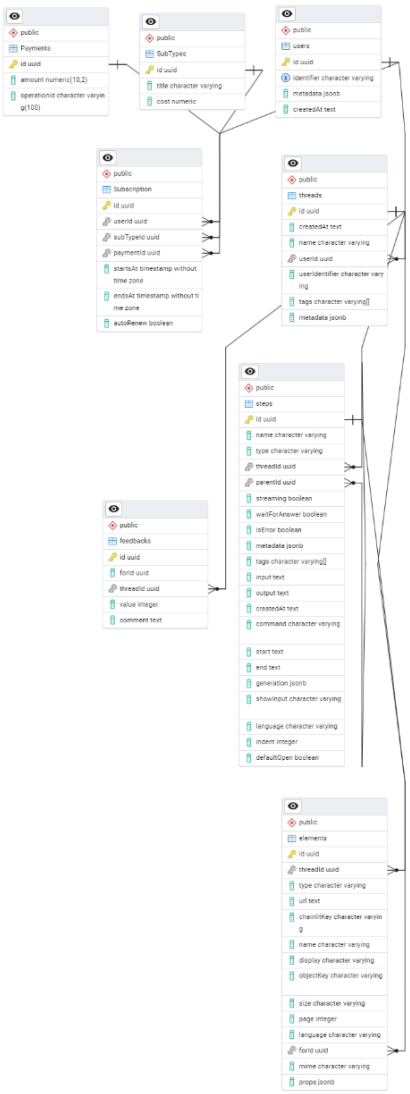

Рисунок 3 – обновлённая схема БД.

Данная схема также обеспечит полноценное и удобное хранение истории чатов, а также, авторизованных пользователей.

3.2 Описание используемых технологий, библиотек и плагинов

В качестве основного языка программирования в данном проекте используется интерпретируемый язык Python с подключенным фреймворком Chainlit, позволяющим разрабатывать чат-ботов. В нём автоматически генерируются страницы с использованием таких языков, как HTML, CSS и JavaScript с библиотекой React, которая в свою очередь позволяет использовать различные анимации.

Помимо фреймворка в продукте используется следующие технологии:

- Prisma Migrate для миграций схемы БД;
- SQLAlchemy – библиотека для настройки общения продукта с базой данных;
- Aiohttp – библиотека для отправки HTTP-запросов;
- Poetry – инструмент для управления зависимости внутри проекта Python;
- Docker – контейнеризация приложения для переноса и развёртывания на выделенном сервере;
- OpenAI – искусственный интеллект;
- PostgreSQL – СУБД;
- Tochka – платёжный шлюз.

Во время работы над продуктом принято решение заменить Prisma Migrate на инструмент миграций Alembic. Данное решение обусловлено совместимостью с библиотекой SQLAlchemy, что сэкономит разработчику время при обновлении схемы базы данных.

3.3 Описание разработанных процедур и функций

В таблице 6 представлена информация о разработанных процедурах и функциях, необходимых для работы приложения, в виде названия и описания деятельности.

Таблица 6 – Описание разработанных процедур и функций

Название	Описание
1	2
get_engine	Создает и возвращает асинхронное подключение к PostgreSQL через SQLAlchemy
get_session	Предоставляет сессию для работы с базой данных
create_tables	Выполняет миграцию и создание всех таблиц в БД
generate_payment_link	Генерирует платежную ссылку через API Точки
generate_sandbox_payment_link	Аналогичная функция для тестового режима
show_sub_status	Отображает текущий статус подписки пользователя
on_chat_start	Инициализирует чат-сессию, настраивает команды
on_message	Основной обработчик входящих сообщений
setup_runnable	Настраивает цепочку обработки запросов с LangChain
parse_repository_info	Парсит информацию о репозитории
open_editor	Открывает sidebar с редактором кода
on_action	Обработчик кнопок действий
oauth_callback	Обработчик OAuth-аутентификации

Ниже представлен код, содержащий в себе метод для создания тестовой ссылки на оплату под названием generate_sandbox_payment_link().

```
async def create_payment_link_sandbox(self, amount: str, uuid: str):
        request_data = {
            "customerCode": str(self.customer_code),
            "amount": amount,
            "purpose": "Подписка deffun",
            "paymentMode": ["sbp", "card"],
            "redirectUrl": self.success_redirect_url,
            "failRedirectUrl": self.failure_redirect_url,
            "consumerId": uuid,
            "merchantId": "123123123123123"
        }

        api_request = {
            "Data": request_data
        }
        print(api_request)
        request_json = json.dumps(api_request)

        headers = {
            "Authorization": f"Bearer working_token",
            "Content-Type": "application/json"
        }

        async with aiohttp.ClientSession() as session:
            async with session.post(f"{test_api_uri}/payments",
data=request_json, headers=headers) as response:
                print(response)
                if response.status == 200:
```

```
                    response_json = await response.json()
                    return response_json.get("Data")
                else:
                    response_text = await response.text()
                    raise Exception(f"Request failed with status
{response.status}: {response_text}")
```

Данная функция генерирует и отсылает запрос на предоставленную ссылку в классе, также берёт некоторые данные из переменных окружения. Далее она вызывается в основном приложении и в последствии бот отправляет эту ссылку в чат к пользователю.

4 Тестирование модулей для приложения

4.1 Выбор стратегии тестирования

В качестве стратегии тестирования программного продукта мной выбрана стратегия методом чёрного ящика, которая фокусируется на проверке функциональности системы без знания внутренней архитектуры.

Выбор такой стратегии обоснован тем, что такой подход не требует доступа к исходному коду программного продукта, поскольку основан на формах входных и выходных данных. Такой вариант стратегии тестирования позволяет оценить работоспособность проектируемой системы с точки зрения конечного пользователя.

На основе такой стратегии достаточно просто составить протоколы тестирования, поскольку известны входные данные, действия для получения результата, ожидаемый, а также записать итоговый фактический результат, что является оценочной характеристикой работоспособности программных компонентов системы. По таким протоколам другие разработчики смогут проводить тестирования при помощи других стратегий, а также автоматизировать процесс тестирования программного продукта.

4.2 Протоколы тестирования

Все протоколы тестирования расположены в таблицах с 7 по 11.

Таблица 7 – Тестирование функции «Генерация кода»

Наименование	Описание
Приоритет	Высокий
Наименование тестирования	Проверка генерации Python-кода по текстовому запросу
Шаги тестирования	1. Открыть чат; 2. Отправить текстовый запрос; 3. Получить сгенерированный код.
Данных тестирования	Текстовое описание: "Напиши функцию для вычисления факториала"
Ожидаемый результат	Корректный Python-код функции вычисления факториала
Фактический результат	Совпадает с ожидаемым

Таблица 8 – Тестирование функции «Анализ GitHub-репозитория»

Наименование	Описание
Приоритет	Высокий
Наименование тестирования	Проверка анализа публичного GitHub-репозитория
Шаги тестирования	1. Открыть чат; 2. Ввести команду /github с ссылкой на репозиторий; 3. Получить анализ кода.
Данных тестирования	Ссылка на тестовый публичный репозиторий с Python-кодом
Ожидаемый результат	Детализированный отчет о структуре репозитория и качестве кода
Фактический результат	Совпадает с ожидаемым

Таблица 9 – Тестирование функции «Оформление подписки»

Наименование	Описание
Приоритет	Высокий
Наименование тестирования	Проверка процесса оформления подписки
Шаги тестирования	1. Ввести команду /purchase; 2. Перейти по платежной ссылке; 3. Выполнить тестовый платеж в sandbox-режиме.
Данных тестирования	Тестовые платежные данные sandbox-режима
Ожидаемый результат	Успешная активация подписки после оплаты
Фактический результат	Совпадает с ожидаемым

Таблица 10 – Тестирование функции «Проверка статуса подписки»

Наименование	Описание
Приоритет	Средний
Наименование тестирования	Проверка отображения информации о подписке
Шаги тестирования	1. Ввести команду /mysub; 2. Получить информацию о текущей подписке.
Данных тестирования	Активный тестовый аккаунт с подпиской
Ожидаемый результат	Корректное отображение типа подписки, сроков действия
Фактический результат	Совпадает с ожидаемым

Таблица 11 – Тестирование функции «Работа с историей запросов»

Наименование	Описание
Приоритет	Средний
Наименование тестирования	Проверка сохранения и отображения истории запросов
Шаги тестирования	1. Выполнить несколько запросов на генерацию кода; 2. Открыть раздел истории; 3. Проверить сохраненные запросы.
Данных тестирования	5 тестовых запросов разного типа
Ожидаемый результат	Все запросы сохранены и доступны для просмотра
Фактический результат	Совпадает с ожидаемым

Заключение

В ходе производственной практики были выполнены все поставленные задачи, а именно:

− была изучена открытая документация к Tochka API;

− фреймворк Chainlit успешно изучен на уровне понимания работы процессов в библиотеке;

− база данных доработана и позволяет удобно и надёжно хранить все данные о подписках.

По итогам проделано работы были приобретены новые умения, которые помогли приобрести больше опыта и знаний для дальнейшего совершенствования в различных разработках, а также в изучении новых технологий.

Библиография

Нормативно-правовые акты:

1 ГОСТ Р 2.105-2019. ЕСКД. Общие требования к текстовым документам. – Москва: Стандартинформ, 2019. – 36 страниц;

2 ГОСТ Р 7.0100-2018. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – Москва: Стандартинформ, 2018. – 128 страниц;

3 ГОСТ 15.016-2016. Техническое задание. – Москва: Стандартинформ, 2017. – 30 страниц.

Электронные ресурсы:

1 Chainlit [Электронный ресурс]: Документация к фреймворку. – URL: https://docs.chainlit.io/get-started/overview (дата обращения: 03.06.2025);

2 Prisma Migrate [Электронный ресурс]: Документация к инструменту – URL: https://www.prisma.io/docs/orm/prisma-migrate (дата обращения: 05.06.2025);

3 Что такое Docker и как он работает [Электронный ресурс]: Базовые знания – URL: https://skillbox.ru/media/code/kak-rabotaet-docker-podrobnyy-gayd-ot-tekhlida/ (Дата обращения 05.06.2025);

4 Быстрый запуск PostgreSQL через Docker Compose [Электронный ресурс]: Инструкция по развёртыванию базы данных в Docker – URL: https://habr.com/ru/articles/823816/ (дата обращения 12.06.2025).