Учреждение образования «БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИТ Кафедра ПИ
Специальность 1-40 01 01 Программное обеспечение информационных технологий
Специализация 1-40 01 01 10 Программирование интернет-приложений

УТВЕРЖДАЮ
Заведующий кафедрой

_____ В.В. Смелов

«____» _____ 2025 г.

ЗАДАНИЕ
на дипломный проект студенту

Точило Олегу Вячеславовичу
(фамилия, имя, отчество)

1. Тема проекта: Web-приложение «GPTranslate» для перевода текста ограниченного объёма с иностранного языка с применением сервиса «g4f»
утверждена приказом по университету от 24 января 2025 г. № 10-С
2. Срок сдачи студентом законченного проекта: 31 мая 2025 г.
3. Требования к программному обеспечению:
– поддерживать роли администратора, модератора, пользователя и гостя;
– гость: регистрироваться, аутентифицироваться;
– пользователь, модератор и администратор: изменять свою учётную запись, просматривать и закрывать открытые сессии;
– пользователь: покупать токены, изменять список исходных и переведённых статей, жалоб, комментариев и конфигураций переводчика;
– модератор: изменять список открытых жалоб, в том числе создавать комментарии к ним;
– администратор: просматривать статистику жалоб на переводы и изменять список моделей переводчика, стилей перевода и пользователей, в том числе создавать пользователей, модераторов и администраторов, изменять пользователей и удалять их.
4. Программная платформа и технологии: Docker 28, Python 3.13, FastAPI 0.115, SQLAlchemy 2, aio-pika 9.5, PostgreSQL 17, Vue.js 3.5, RabbitMQ 4.1.
5. Содержание расчетно-пояснительной записки:
1) реферат;
2) содержание;
3) введение;
4) раздел 1: постановка задачи и аналитический обзор аналогичных решений;
5) раздел 2: проектирование веб-приложения;
6) раздел 3: разработка веб-приложения;
7) раздел 4: тестирование веб-приложения;
8) раздел 5: руководство по эксплуатации;
9) раздел 6: технико-экономическое обоснование проекта;
10) заключение;
11) список использованных источников;

12) приложения и графическая часть;

6. Перечень графического материала (с точным указанием обязательных чертежей):

1) диаграмма вариантов использования;

2) диаграмма развертывания;

3) логическая схема базы данных;

4) блок-схема алгоритма перевода статьи;

5) диаграмма последовательности;

6) скриншот работы программы.

7. Консультанты по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант
Технико-экономическое обоснование проекта	*Познякова Л.С.*

8. Дата выдачи задания: 24 января 2025 г.

Руководитель_____ Белодед Н.И.
(подпись)

Задание принял к исполнению_____Точило О.В.
(подпись)

Календарный план

№ п/п	Наименование этапов дипломного проекта	Срок выполнения этапов проекта	Примечание
1	*Патентный поиск, обзор аналогов, прототипов, анализ предметной области*	24.01.2025 г.	
2	*Постановка задачи и разработка функциональных требований*	25.01.2025 г.	
3	*Проектирование архитектуры программы/системы/модуля (разработка структуры системы, алгоритмов и схемы данных)*	29.01.2025 г.	
4	*Реализация программы/системы/модуля*	15.02.2025 г.	
5	*Тестирование и отладка*	30.04.2025 г.	
6	*Выполнение расчетов экономического раздела*	07.05.2025 г.	
7	*Оформление пояснительной записки и графического материала*	24.05.2025 г.	
8	*Итоговая проверка готовности дипломного проекта на заседании рабочей комиссии кафедры и допуск к защите в ГЭК*	31.05.2025 г.	
9	*Рецензирование дипломного проекта*	до 11.06.2025 г.	
10	*Защита дипломного проекта*	с 16.06.2025 г.	

Дипломник _____ Руководитель проекта_____
(подпись) (подпись)

Реферат

Пояснительная записка содержит 52 страницы, 6 рисунков, 24 таблицы, 14 источников, 3 приложения.

WEB-ПРИЛОЖЕНИЕ, FASTAPI, POSTGRESQL, VUE.JS, DOCKER

Объектом дипломного проекта является web-приложение для перевода текста ограниченного объёма с иностранного языка с применением сервиса «g4f».

Основной целью дипломного проекта является разработка веб-приложения для перевода текста ограниченного объёма с иностранного языка с применением сервиса «g4f». В разработке дипломного проекта была использована платформа FastAPI, язык программирования Python, технология Vue.js, протокол обмена данными HTTP.

Пояснительная записка состоит из введения, семи разделов и заключения.

В первом разделе описана литература, использовавшаяся при разработке дипломного проекта.

Во втором разделе описаны цель и задачи дипломного проекта, обзор аналогов и обзор средств разработки.

В третьем разделе представлены архитектура web-приложения, проектирование и структура таблиц базы данных.

Четвёртый раздел посвящен разработке web-приложения.

Пятый раздел посвящен тестированию web-приложения.

В шестом разделе приведено руководство программиста.

В седьмом разделе приводится расчет экономических параметров и себестоимость программного продукта.

В заключении представлены итоги дипломного проекта и задачи, которые были решены в ходе разработки программного средства.

Содержание

Введение

Тема проекта «Web-приложение «GPTranslate» для перевода текста ограниченного объёма с иностранного языка с применением сервиса «g4f»» означает, что результатом выполнения проекта является web-приложение, позволяющее пользователям переводить текст на исходном языке в текст на другом языке (например, с польского языка на немецкий) с использованием внешнего сервиса «g4f», предоставляющего доступ к нейронным сетям. Доступ к внешнему сервису осуществляется по его API. Ограниченный объём текста означает, что в web-приложении установлено ограничение на максимальную длину исходного текста.

Основная цель проекта – повысить эффективность и увеличить скорость перевода текста за счёт использования внешних сервисов, а также предоставить пользователям механизм обратной связи для улучшения сервиса.

Для достижения поставленной цели в рамках дипломного проекта были сформулированы следующие задачи: в разделе 1 провести анализ литературных источников; в разделе 2 провести анализ существующих сервисов перевода и выявить их преимущества и недостатки; в разделе 3 разработать архитектуру и структуру приложения, включая выбор технологий и структуру базы данных; в разделе 4 описать программную разработку с реализацией ключевых функций; в разделе 5 провести тестирование функционала и производительности; в разделе 6 подготовить техническую документацию и инструкции по развёртыванию приложения; в разделе 7 провести технико-экономический анализ проекта.

Целевая аудитория приложения включает широкий спектр пользователей: от профессиональных переводчиков и сотрудников международных компаний до владельцев web-сайтов и блогеров, нуждающихся в качественном и быстром переводе своих материалов.

Для реализации web-приложения было решено использовать язык программирования Python и фреймворк FastAPI из-за высокой гибкости, распространённости и простоты данных технологий.

1 Обзор литературных источников

1.1 Документация Python

Официальная документация Python представляет собой комплексную систему материалов, организованную по принципу ”от простого к сложному”. Она выделяется среди других языков своей полнотой и доступностью изложения. Она включает в себя следующие ключевые разделы: учебник для начинающих (пошаговое введение в синтаксис и концепции языка, построенное по принципу ”сценарной документации что позволяет новичкам быстро добиваться нужных результатов по заранее расписанным шагам), справочник по библиотеке (исчерпывающее описание стандартной библиотеки, включающее подробные примеры использования каждого модуля), справочник по языку (детальное описание синтаксиса и семантики языка, включая формальные определения и спецификации), а также руководство по расширению и внедрению Python (информация для разработчиков, создающих модули на языках программирования C/C++ или встраивающих интерпретатор Python в другие приложения).

Документация Python решает следующие проблемы: обеспечение единого понимания языка среди разработчиков, стандартизация использования встроенных возможностей и библиотек, снижение порога вхождения для новых разработчиков, выступление в качестве авторитетного источника при разрешении технических споров и неоднозначностей.

1.2 Документация FastAPI

Документация фреймворка FastAPI предоставляет полную информацию для использования данной технологии. Она содержит руководство пользователя для быстрого начала работы, пошаговые инструкции к различным аспектам фреймворка с пояснениями каждого шага и примерами для различных версий смежных технологий (например, с использованием typing.List в Python 3.8 и list в Python 3.9 и позднее), а также описание взаимодействия с прочими технологиями (например, Starlette и Pydantic).

Кроме того, документация предоставляет доступ к описанию типов, классов и функций с их подробным описанием, что позволяет ускорить разработку приложений. Также документация предоставляет рекомендации по развёртыванию приложений в продуктовых окружениях и настройке в случае нахождения за прокси.

1.3 Документация Pydantic

Документация библиотеки валидации Pydantic организована как интерактивное руководство, сочетающее теоретические объяснения с практическими примерами. Она выделяется среди других библиотек своим подходом ”проблема - решение где каждый раздел начинается с описания конкретной задачи разработки, за чем следует описание её решения.

Так же, как документация FastAPI, документация Pydantic содержит руководство для начинающих для быстрого начала работы, руководства по основным аспектам библиотеки, а также раздел с описанием функций и классов, реализуемых библиотекой. Документация позволяет узнать, каким образом проектировать и реализовывать надёжные API с сериализацией и валидацией.

1.4 Документация PostgreSQL

Документация СУБД PostgreSQL представляет исчерпывающее руководство по ключевым аспектам применения данной технологии, а также инструкции по развёртыванию и использованию. Кроме того, документация содержит полное описание диалекта PostgreSQL с примерами использования, что позволяет лучше изучить его для полноценного использования СУБД.

Кроме того, документация описывает внутренне устройство СУБД, включая индексы, системы обеспечения согласованности, доступности, репликации, партиционирования и шардирования, и заметки к выпуску каждой новой версии, включая мелкие и крупные изменения.

Документация PostgreSQL предоставляет исчерпывающую информацию о возможностях системы, облегчает процесс настройки и оптимизации производительности, помогает в проектировании схемы данных и разработке сложных запросов, а также служит справочником при решении проблем и отладке ошибок.

2 Постановка задачи и обзор аналогичных решений

2.1 Постановка задачи

Web-приложение должно позволять пользователю зарегистрироваться по адресу имени, электронной почты и паролю, подтвердить адрес электронной почты по ссылке из электронного письма, аутентифицироваться по адресу электронной почты и паролю, создать конфигурацию переводчика, загрузить исходную статью с введённым с клавиатуры текстом или при помощи загруженного текстового файла, выполнить перевод, используя созданную ранее конфигурацию переводчика, получить уведомление о завершении перевода, просмотреть переведённую статью. В случае, если перевод не удовлетворяет пользователя, web-приложение должно позволять пользователю создать жалобу на перевод, а модератору – просмотреть жалобу и удовлетворить либо отклонить её.

2.2 Обзор аналогичных решений

Одним из самых популярных сервисов по переводу текста с одного языка на другой является DeepL. Он предоставляет возможность перевода текста между различными языками, распознавание голоса, загрузку файлов и пересказ текста. Внешний вид страницы сервиса представлен на рисунке 2.1.

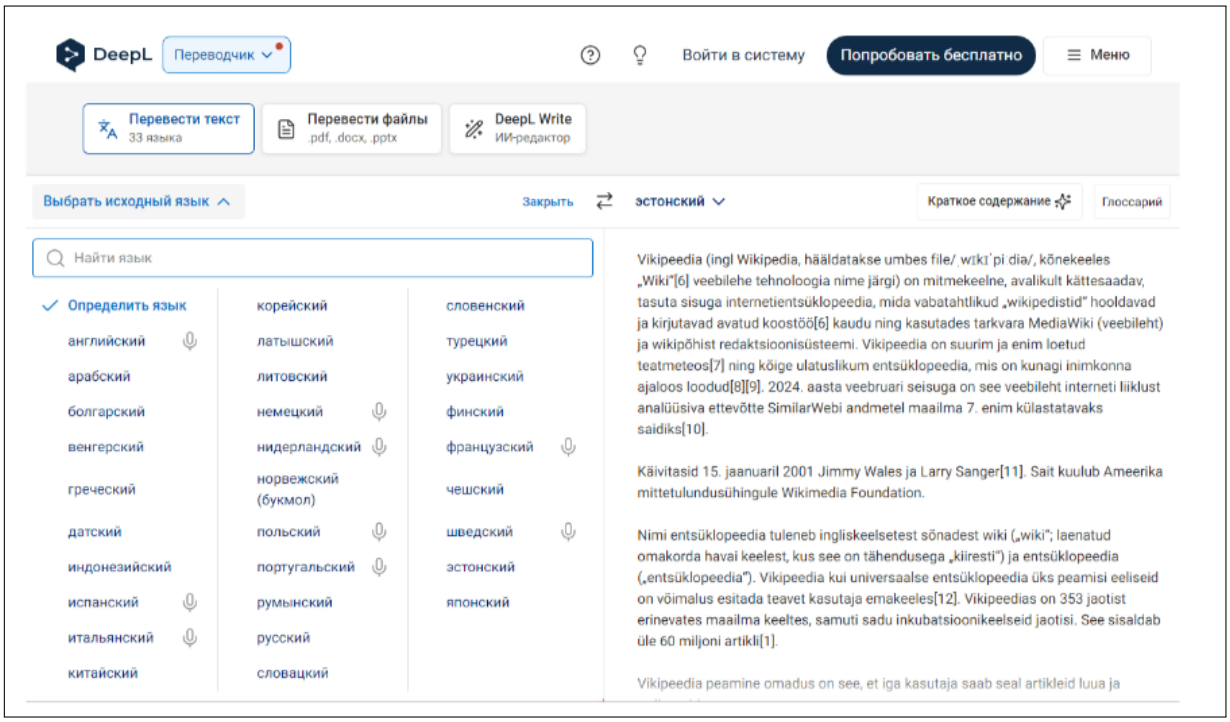

Рисунок 2.1 – Страница сервиса DeepL

Данный сервис применяет нейронные сети для перевода текста, что положительно сказывается на качестве перевода текста.

В качестве второго аналогичного решения был рассмотрен сервис Google Translate. Внешний вид страницы данного сервиса представлен на странице 2.2.

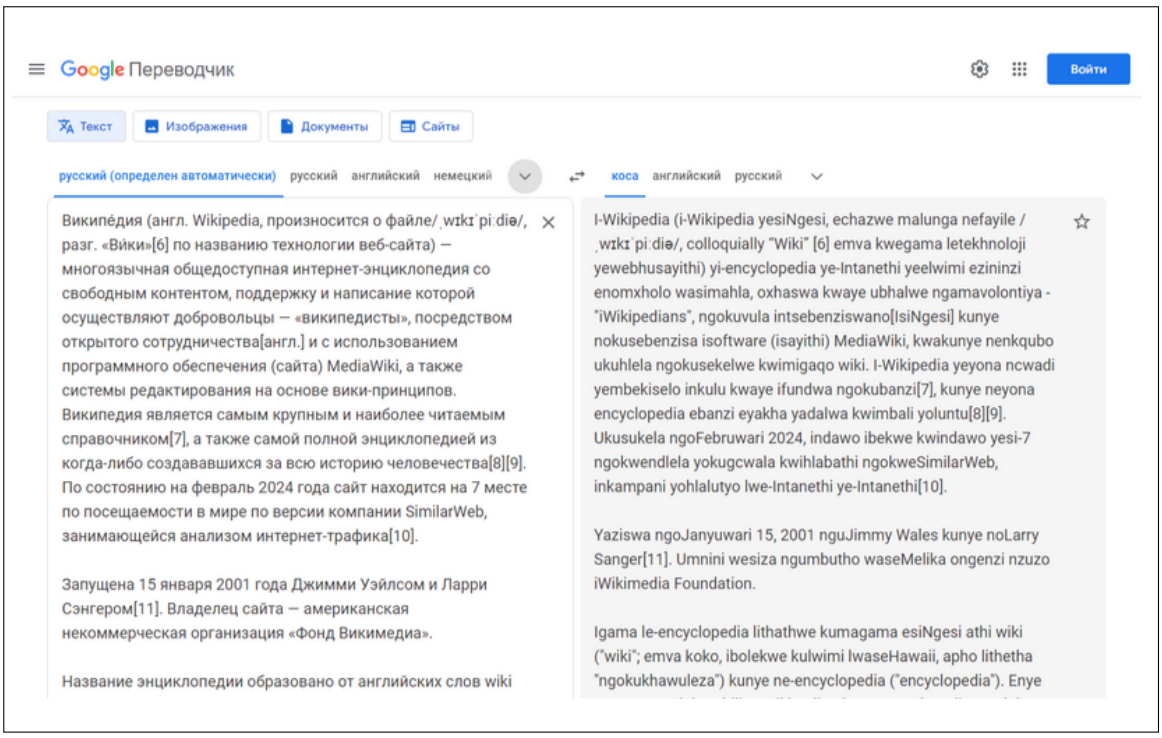

Рисунок 2.2 – Страница сервиса Google Translate

Это один из самых известных и широко используемых сервисов машинного перевода. Он также предоставляет возможность перевода текстовых файлов, а также более широкий выбор языков по сравнению с DeepL.

В качестве третьего аналогичного решения был рассмотрен сервис Wordvice. Внешний вид его страницы представлен на рисунке 2.3.

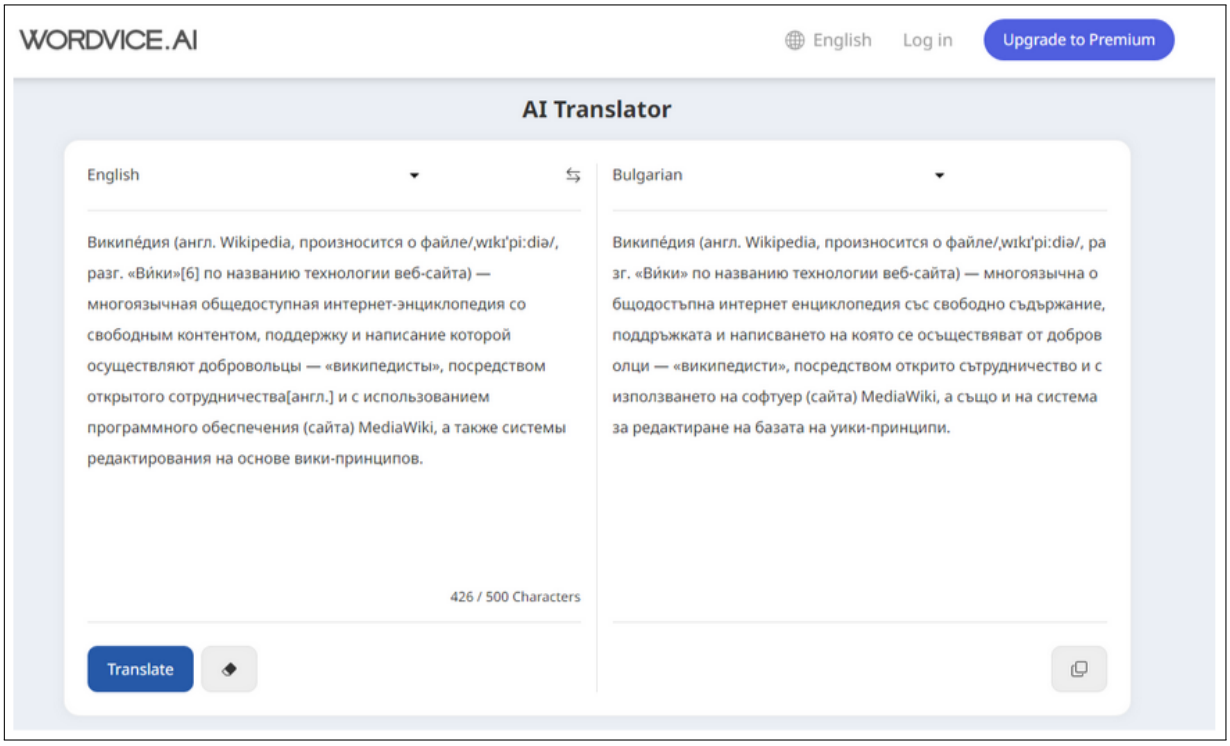

Рисунок 2.3 – Страница сервиса Wordvice

Он также использует нейронные сети для перевода текста, предоставляет ин-

теграцию с Microsoft Word и услуги обобщения и перефразирования текста при помощи искусственного интеллекта, а также поддерживает множество языков.

2.3 Выводы

1. Был рассмотрен сценарий использования web-приложения, что позволяет выделить ключевые функции web-приложения.
2. Анализ существующих решений в области перевода текста выявил как преимущества, так и недостатки сервисов-конкурентов. Сервисы DeepL, Google Translate и Wordvice предоставляют базовый функционал, включая перевод текста, загрузку текстовых документов и выбор языков. Среди ключевых недостатков отмечается отсутствие персонализации, невозможность выбора средства выполнения перевода и его стиля, а также отсутствие жалоб на переводы.

3 Проектирование web-приложения

3.1 Функциональность web-приложения

Функциональные возможности web-приложения представлены в диаграмме вариантов использования, представленной на рисунке 3.1.

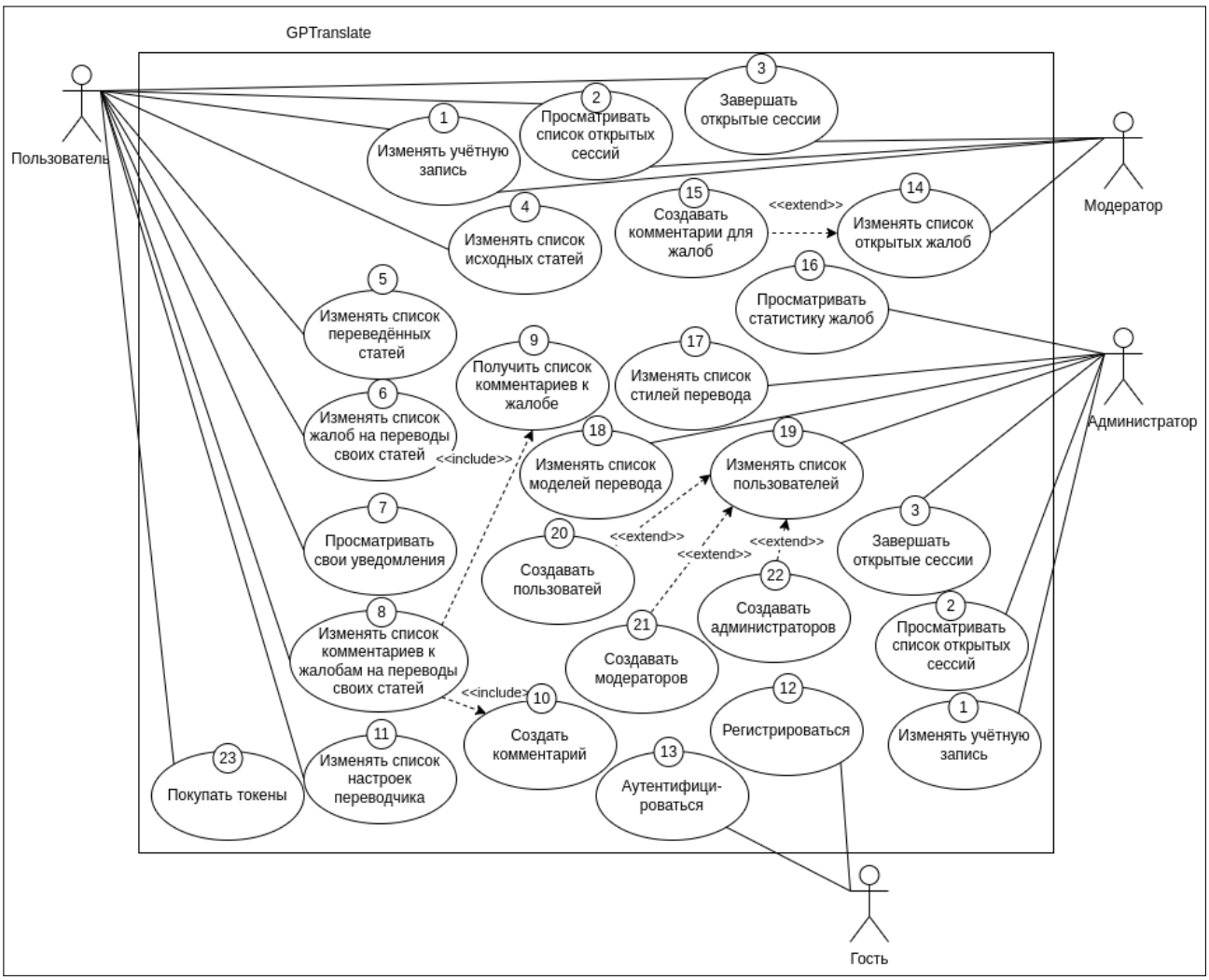

Рисунок 3.1 – Диаграмма вариантов использования web-приложения

Перечень ролей и их назначение приведены в таблице 3.1.

Таблица 3.1 – Назначение ролей пользователей в web-приложении

Роль	Назначение
Гость	Регистрация и аутентификация
Пользователь	Загрузка и запуск перевода статей, получение переводов, создание жалоб на переводы своих статей
Модератор	Рассмотрение жалоб на переводы
Администратор	Управление пользователями, запросами перевода, моделями перевода

Функциональные возможности пользователя с ролью «Гость» представлены в таблице 3.2.

Таблица 3.2 – Функциональные возможности пользователя с ролью «Гость»

Вариант использования	Пояснение
12 Регистрироваться	Гость может создать учётную запись при помощи электронной почты и пароля или OAuth 2.0-провайдера
13 Аутентифицироваться	Гость может аутентифицироваться при помощи электронной почты и пароля или OAuth 2.0-провайдера

После аутентификации гость становится либо пользователем, либо модератором, либо администратором. По этой причине пользователю недоступна аутентификация. Функциональные возможности пользователя с ролью «Пользователь» представлены в таблице 3.3.

Таблица 3.3 – Функциональные возможности пользователя с ролью «Пользователь»

Вариант использования	Пояснение
1 Изменять учётную запись	Изменять своё имя и пароль
2 Просматривать список открытых сессий	Получать список открытых сессий
3 Завершать открытые сессии	Блокировать доступ для всех открытых сессий
4 Изменять список исходных статей	Загружать из файла или вводить с клавиатуры исходные статьи, получать список исходных статей, изменять содержимое исходных статей, удалять их
5 Изменять список переведённых статей	Запускать перевод исходных статей, получать их список, оставлять оценку переводам статей, удалять переводы статей
6 Изменять список жалоб на переводы своих статей	Создавать жалобы на переводы своих статей, получать их список, закрывать открытые жалобы на переводы своих статей
7 Просматривать свои уведомления	Получать список непрочитанных уведомлений

Продолжение таблицы 3.3

8 Изменять список комментариев к жалобам на переводы своих статей	Получать список комментариев, создавать комментарии к открытым жалобам на переводы своих статей
9 Получить список комментариев к жалобе	Получить список комментариев к одной из своих жалоб
10 Создать комментарий	Создать комментарий к одной из своих жалоб
11 Изменять список настроек переводчика	Получать список своих конфигураций, создавать новые, обновлять и удалять существующие конфигурации
23 Покупать токены	Совершать покупки токенов через внешнюю систему оплаты для последующего использования в переводах

Модератор может рассматривать жалобы пользователей. Его функциональные возможности представлен в таблице 3.4.

Таблица 3.4 – Функциональные возможности пользователя с ролью «Модератор»

Вариант использования	Пояснение
1 Изменять учётную запись	Изменять своё имя и пароль
2 Просматривать список открытых сессий	Получать список открытых сессий
3 Завершать открытые сессии	Блокировать доступ для всех открытых сессий
14 Изменять список открытых жалоб	Получать список открытых жалоб на переводы, получать списки комментариев и создавать новые комментарии к ним, принимать или отклонять жалобы
15 Создавать комментарии для жалоб	Создавать комментарии для открытой жалобы

Функциональные возможности администратора представлен в таблице 3.5.

Таблица 3.5 – Функциональные возможности пользователя с ролью «Администратор»

Вариант использования	Пояснение
1 Изменять учётную запись	Изменять своё имя и пароль
2 Просматривать список открытых сессий	Получать список открытых сессий

Продолжение таблицы 3.5

3 Завершать открытые сессии	Блокировать доступ для всех открытых сессий
16 Просматривать статистику жалоб	Получать данные о том, какая часть переводов при помощи каждой модели получает жалобы и какая их доля удовлетворяется модераторами
17 Изменять список стилей перевода	Создавать новые стили, обновлять и удалять существующие
18 Изменять список моделей перевода	Добавлять информацию о новых моделях, изменять и удалять существующие записи
19 Изменять список пользователей	Получать список пользователей, создавать новых, изменять и удалять существующих
20 Создавать пользователей	Создавать объекты пользователей с ролью пользователя
21 Создавать модераторов	Создавать объекты пользователей с ролью модератора
22 Создавать администраторов	Создавать объекты пользователей с ролью администратора

Таким образом, пользователю доступны базовые операции, такие как операции над статьями и настройками перевода, модераторы могут управлять жалобами, а администраторы – управлять пользователями, моделями, запросами перевода и просматривать статистику жалоб на переводы.

3.2 Структура базы данных

Согласно схеме вариантов использования была создана база данных. Её структура представлена на рисунке 3.2.

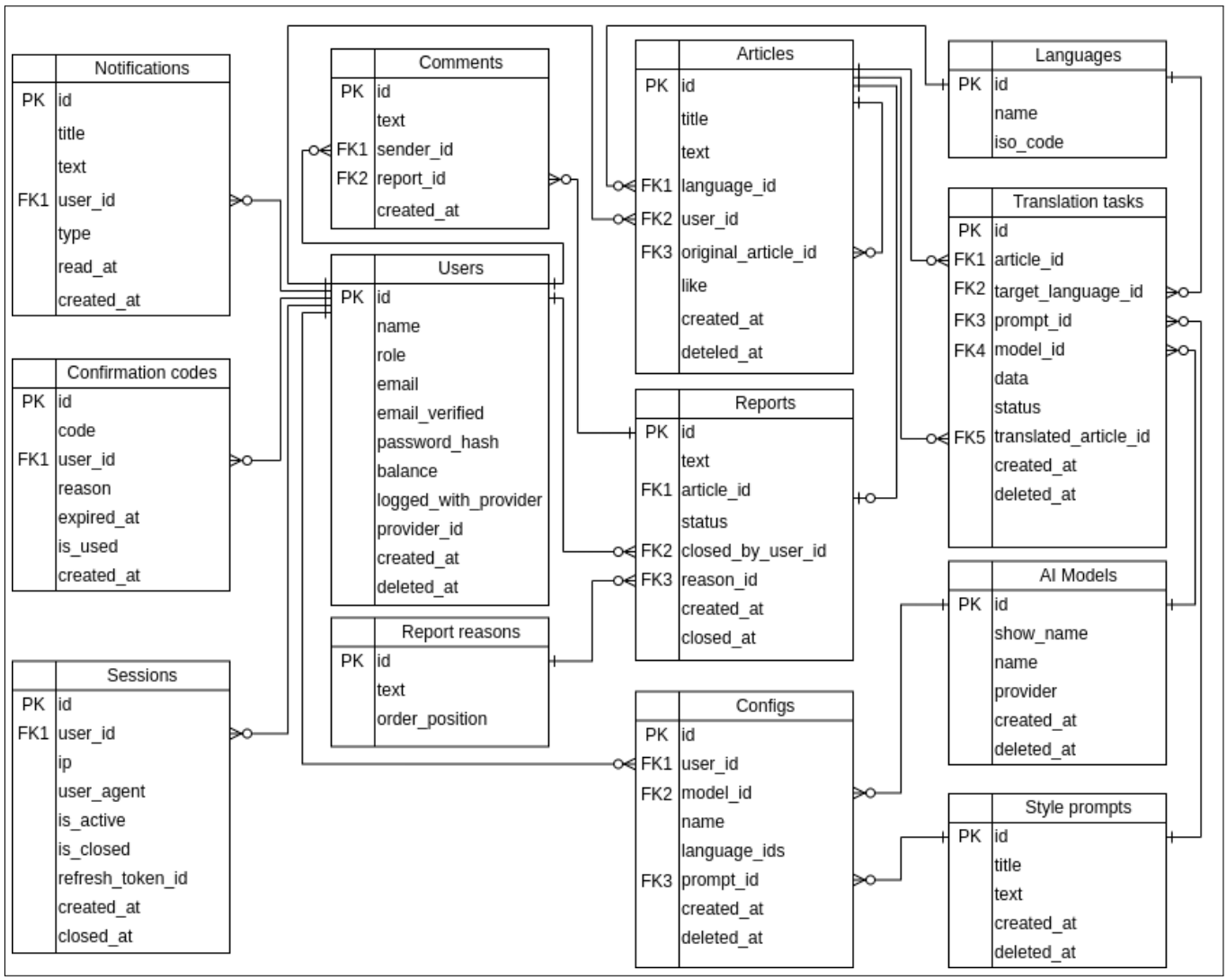

Рисунок 3.2 – Логическая схема базы данных

База данных содержит тринадцать таблиц, хранящих информацию о пользователях, сессиях, статьях и прочих данных. Типы данных были выбраны согласно [1]. Назначение таблиц базы данных представлено в таблице 3.6.

Таблица 3.6 – Назначение таблиц базы данных

Таблица	Назначение
Users	Хранит информацию о пользователях (имя, адрес электронной почты и хеш пароля для аутентификации и так далее)
Sessions	Хранит информацию о сессиях пользователей (идентификатор пользователя, флаг активности, время создания и так далее)
Confirmation_codes	Хранит информацию о кодах подтверждения адреса электронной почты и сброса пароля

Продолжение таблицы 3.6

Languages	Хранит информацию о доступных для перевода языках (название, ISO код)
Articles	Хранит информацию о статьях (заголовок, текст, идентификатор пользователя и так далее)
Report_reasons	Хранит информацию о доступных причинах для жалобы на перевод статьи (текст, позиция в списке для сортировки)
Reports	Хранит информацию о жалобах на переводы статей (идентификатор статьи, текст, идентификатор, причина и так далее)
Report_comments	Хранит информацию о комментариях к жалобам на переводы статей (текст, идентификатор пользователя, идентификатор жалобы, дата и время создания)
Style_prompts	Хранит информацию о запросах перевода с разными стилями (название, текст и так далее)
AI_Models	Хранит информацию о моделях искусственного интеллекта, использующихся для перевода (название, поставщик и так далее)
Configs	Хранит информацию о конфигурациях переводчика, которые могут использоваться пользователями для упрощения запуска перевода своих статей (идентификаторы запроса перевода, модели, языков и так далее)
Translation_tasks	Хранит информацию о задачах перевода, которые считываются отдельным процессом и выполняются им (идентификаторы статьи, модели, исходного и конечного языков, статус и так далее)
Notifications	Хранит информацию об уведомлениях пользователей (идентификатор пользователя, текст, тип уведомления и так далее)

Описание столбцов таблицы Users представлено в таблице 3.7.

Таблица 3.7 – Описание таблицы Users

Название столбца	Тип данных	Описание
id	uuid	Идентификатор пользователя, первичный ключ
name	varchar (20)	Имя пользователя
email	varchar (50)	Адрес электронной почты пользователя
email_verified	boolean	Флаг, указывающий, был ли подтверждён адрес электронной почты пользователя
password_hash	varchar (60)	Хеш пароля соискателя
role	enum user_role	Роль пользователя (пользователь, модератор, администратор)
logged_with_provider	varchar	Название провайдера OAuth 2.0, использовавшегося для регистрации
provider_id	varchar	Идентификатор пользователя, полученный от провайдера OAuth при регистрации
created_at	timestamp without timezone	Дата и время создания пользователя без часового пояса
deleted_at	timestamp without timezone	Дата и время удаления пользователя без часового пояса

Таблица Sessions хранит данные о сессиях пользователей. Описание её столбцов представлено в таблице 3.8.

Таблица 3.8 – Описание таблицы Sessions

Название столбца	Тип данных	Описание
id	uuid	Идентификатор сессии, первичный ключ

Продолжение таблицы 3.8

user_id	uuid	Идентификатор пользователя, который создал данную сессию, внешний ключ
ip	varchar (15)	IPv4 адрес узла, из которого была открыта сессия
user_agent	varchar (100)	User agent клиента (например, браузера)
is_closed	boolean	Флаг, указывающий, была ли сессия закрыта
refresh_token_id	uuid	Идентификатор refresh токена, связанного с данной сессией
created_at	timestamp without timezone	Дата и время создания сессии без часового пояса
closed_at	timestamp without timezone	Дата и время закрытия сессии без часового пояса

Описание столбцов таблицы Confirmation_codes представлено в таблице 3.9.

Таблица 3.9 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	integer	Идентификатор кода, первичный ключ
code	varchar	Строковое значение кода
reason	enum confirmationtype	Тип кода (подтверждение адреса электронной почты, сброс пароля)
user_id	uuid	Идентификатор пользователя, для которого предназначен данный код подтверждения, внешний ключ
expired_at	timestamp without timezone	Временная отметка, после которой код будет считаться истёкшим

Продолжение таблицы 3.9

is_used	boolean	Флаг, указывающий, был ли код использован
created_at	timestamp without timezone	Дата и время создания кода без часового пояса

Таблица Languages хранит информацию о языках, доступных для перевода. Описание её столбцов представлено в таблице 3.10.

Таблица 3.10 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	integer	Идентификатор языка, первичный ключ
name	varchar	Отображаемое название языка
iso_code	varchar	ISO код языка

Таблица Articles хранит информацию об исходных и переведённых статьях. Описание её столбцов представлено в таблице 3.11.

Таблица 3.11 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	uuid	Идентификатор статьи, первичный ключ
title	varchar (50)	Название статьи
text	text	Текст статьи
user_id	uuid	Идентификатор пользователя, которому принадлежит статья, внешний ключ
language_id	integer	Идентификатор языка статьи, внешний ключ
original_article_id	uuid	Идентификатор статьи, переводом которой является данная статья, внешний ключ
like	boolean	Флаг, указывающий, какую оценку пользователь поставил переводу (положительную, отрицательную, не поставил оценку)

Продолжение таблицы 3.11

created_at	timestamp without timezone	Дата и время создания статьи без часового пояса
deleted_at	timestamp without timezone	Дата и время удаления статьи без часового пояса

Описание столбцов таблицы Report_reasons представлено в таблице 3.12.

Таблица 3.12 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	integer	Идентификатор причины, первичный ключ
text	varchar	Текст причины
order_position	integer	Положение причины в списке при сортировке

Описание столбцов таблицы Reports представлено в таблице 3.13.

Таблица 3.13 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	uuid	Идентификатор жалобы, первичный ключ
text	varchar (1024)	Текст жалобы
article_id	uuid	Идентификатор статьи, на которую была оставлена жалоба, внешний ключ
status	enum reportstatus	Статус жалобы (открыта, закрыта пользователем, отклонена, удовлетворена)
closed_by_user_id	uuid	Идентификатор пользователя, закрывшего жалобу (пользователь, которому принадлежит статья или модератор), внешний ключ
reason_id	int	Идентификатор причины, по которой была оставлена жалоба, внешний ключ

Продолжение таблицы 3.13

created_at	timestamp without timezone	Дата и время создания жалобы без часового пояса
closed_at	timestamp without timezone	Дата и время закрытия жалобы без часового пояса

Описание столбцов таблицы Report_comments представлено в таблице 3.14.

Таблица 3.14 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	uuid	Идентификатор комментария, первичный ключ
text	varchar (100)	Текст комментария
sender_id	uuid	Идентификатор пользователя, оставившего комментарий, внешний ключ
report_id	uuid	Идентификатор жалобы, к которой был оставлен комментарий, внешний ключ
created_at	timestamp without timezone	Дата и время создания комментария без часового пояса

Описание столбцов таблицы Style_prompts представлено в таблице 3.15.

Таблица 3.15 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	integer	Идентификатор запроса, первичный ключ
title	varchar (20)	Название запроса
text	varchar	Текст запроса
created_at	timestamp without timezone	Дата и время создания запроса без часового пояса
deleted_at	timestamp without timezone	Дата и время удаления запроса без часового пояса

Описание столбцов таблицы AI_Models представлено в таблице 3.16.

Таблица 3.16 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	integer	Идентификатор модели, первичный ключ
show_name	varchar (50)	Отображаемое название модели
name	varchar	Название модели
provider	varchar	Поставщик модели
created_at	timestamp without timezone	Дата и время создания записи о модели без часового пояса
deleted_at	timestamp without timezone	Дата и время удаления записи о модели без часового пояса

Таблица Configs хранит информацию о конфигурациях переводчика. Описание её столбцов представлено в таблице 3.17.

Таблица 3.17 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	integer	Идентификатор конфигурации, первичный ключ
name	varchar (20)	Название конфигурации
user_id	uuid	Идентификатор пользователя, создавшего конфигурацию, внешний ключ
prompt_id	integer	Идентификатор запроса перевода, внешний ключ
language_ids	integer []	Идентификаторы языков перевода
model_id	integer	Идентификатор модели перевода, внешний ключ
created_at	timestamp without timezone	Дата и время создания конфигурации без часового пояса
deleted_at	timestamp without timezone	Дата и время удаления конфигурации без часового пояса

Таблица Translation_tasks хранит информацию о задачах перевода. Данная информация используется для определения текста исходной статьи, конечного языка и так далее. Описание столбцов таблицы представлено в таблице 3.18.

Таблица 3.18 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	uuid	Идентификатор задачи, первичный ключ
article_id	uuid	Идентификатор исходной статьи, внешний ключ
target_language_id	integer	Идентификатор конечного языка, внешний ключ
prompt_id	integer	Идентификатор запроса перевода, внешний ключ
model_id	integer	Идентификатор модели перевода, внешний ключ
status	enum translationtaskstatus	Статус задачи (создана, в процессе выполнения, завершена успешно, завершена с ошибкой)
data	jsonb	Дополнительная информация о задаче (текст ошибки)
translated_article_id	uuid	Идентификатор переведённой статьи, внешний ключ
created_at	timestamp without timezone	Дата и время создания задачи без часового пояса
deleted_at	timestamp without timezone	Дата и время удаления задачи без часового пояса

Описание столбцов таблицы Notifications представлено в таблице 3.19.

Таблица 3.19 – Назначение таблиц базы данных

Название столбца	Тип данных	Описание
id	uuid	Идентификатор уведомления, первичный ключ
title	varchar	Заголовок уведомления
text	varchar	Текст уведомления
user_id	uuid	Идентификатор пользователя, которому предназначено уведомление, внешний ключ
type	enum notificationtype	Тип уведомления (информационное, предупреждение, ошибка)
created_at	timestamp without timezone	Дата и время создания записи о модели без часового пояса
read_at	timestamp without timezone	Дата и время удаления записи о модели без часового пояса

Назначение связей приведено в таблице 3.20.

Таблица 3.20 – Назначение таблиц базы данных

Связь	Назначение
Users.id – Notifications.user_id	Идентификатор пользователя, которому адресовано уведомление
Users.id – Confirmation_codes. user_id	Идентификатор пользователя, которому предназначен код подтверждения
Users.id – Sessions.user_id	Идентификатор пользователя, который создал сессию
Users.id – Articles.user_id	Идентификатор пользователя, который загрузил статью или запустил перевод исходной статьи
Users.id – Configs.user_id	Идентификатор пользователя, которому принадлежит конфигурация переводчика
Users.id – Commens.sender_id	Идентификатор пользователя, отправившего комментарий

Продолжение таблицы 3.20

Users.id – Reports. closed_by_user_id	Идентификатор пользователя, закрывшего жалобу (создавшего её пользователя или любого модератора)
Report_reasons.id – Reports.reason_id	Идентификатор причины, по которой была создана жалоба на перевод статьи
Articles.id – Articles. original_article_id	Идентификатор исходной статьи, из которой был создан перевод
Articles.id – Translation_tasks. article_id	Идентификатор статьи, которую необходимо перевести
Articles.id – Translation_tasks. translated_article_id	Идентификатор перевода статьи
Articles.id – Reports.article_id	Идентификатор перевода, на который была создана жалоба
Languages.id – Articles.language_id	Идентификатор языка статьи
Languages.id – Translation_tasks. target_language_id	Идентификатор конечного языка, на который необходимо перевести статью
Reports.id – Comments.report_id	Идентификатор жалобы, под которой был оставлен комментарий
AI_Models.id – Translation_tasks. model_id	Идентификатор записи о модели искусственного интеллекта, которая используется для перевода статьи
AI_Models.id – Configs.model_id	Идентификатор записи о модели искусственного интеллекта
Style_prompts.id – Translation_tasks. prompt_id	Идентификатор запроса перевода, который используется для перевода статьи
Style_prompts.id – Configs.prompt_id	Идентификатор запроса перевода

Таким образом, была спроектирована база данных для долговременного хранения информации web-приложения.

3.3 Архитектура web-приложения

Для обеспечения вспомогательных функций web-приложения (отправка почты, выполнение перевода, отправка уведомлений между компонентами системы и так далее) используются дополнительные компоненты.

Для запуска многоконтейнерных Docker-приложений используется инструмент Docker Compose. Он управляет набором контейнеров, в которых работают прочие компоненты web-приложения.

Для хранения данных используется СУБД PostgreSQL 17.

Для обслуживания web-приложение и предоставления доступа к скомпилированному пакету фронтэнд-приложения, созданному с использованием Vue.js, используется web-сервер Nginx.

Для асинхронного обмена сообщениями между компонентами системы используется брокер сообщений RabbitMQ.

Для обработки сообщений, передаваемых через RabbitMQ, используются два процесса-подписчика. Они принимают сообщения из очереди и обрабатывают поступившие команды, такие как перевод статьи и отправка электронной почты для подтверждения регистрации или сброса пароля.

Для быстрого доступа к данным, которые часто используются, например, идентификаторам закрытых сессий, и для передачи уведомлений пользователю используется in-memory база данных Redis.

Архитектура web-приложения представлена на рисунке 3.3.

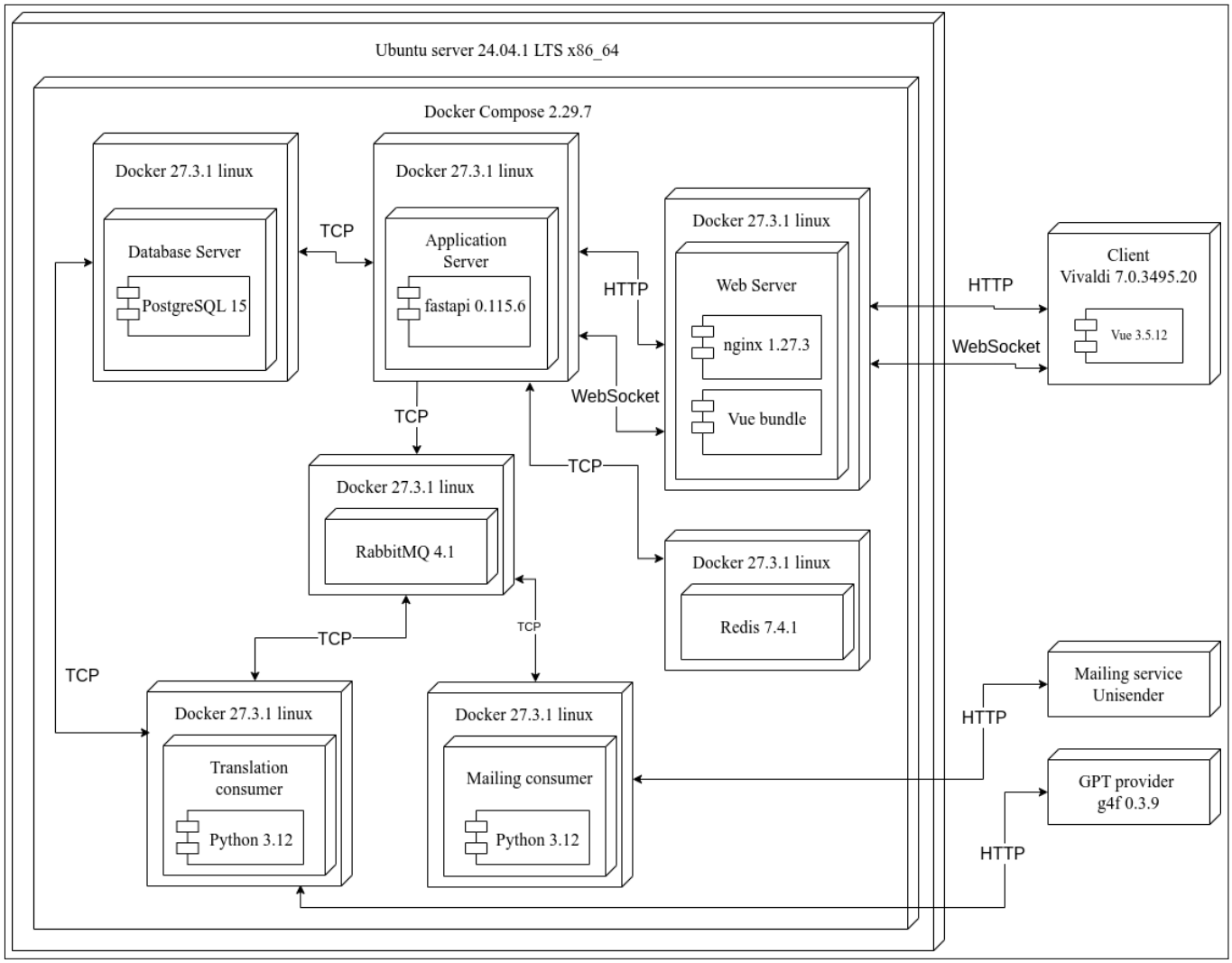

Рисунок 3.3 – Архитектура web-приложения

Пояснение назначения каждого элемента web-приложения на архитектурной диаграмме представлено в таблице 3.21.

Таблица 3.21 – Назначение элементов архитектурной схемы web-приложения

Элемент	Назначение
Web Server (nginx)	Принимать запросы клиента, обеспечивать работу HTTPS, предоставлять статические файлы фронтэнд-части web-приложения
Database Server (PostgreSQL)	Хранить данные, которые должны храниться длительное время
RabbitMQ	Обеспечивать обмен сообщениями между компонентами web-приложения
Application Server	Обрабатывать запросы пользователя
Translation consumer	Переводить статьи при помощи внешнего сервиса
Mailing consumer	Отправлять электронные письма при помощи внешнего сервиса
Redis	Хранить данные с маленьким сроком жизни, выступать транспортом для отправки уведомлений о завершении перевода статей
GPT provider	Переводить тексты по запросу
Mailing service	Отправлять электронные письма по запросу
Client (Vivaldi)	Отображать фронтэнд-часть web-приложения, отправлять запросы пользователя, отображать ответы сервера

Таким образом, web-приложение состоит из различных компонентов, каждый из которых выполняет собственные функции.

3.4 Выводы

1. Была рассмотрена функциональность web-приложения «GPTranslate» для всех ролей: гостя, пользователя, модератора и администратора. Гостям доступна регистрация и аутентификация. Пользователи могут загружать статьи, переводить их, а также управлять своими конфигурациями переводчика и оставлять жалобы на переведённые статьи. Модераторы рассматривают жалобы, а администраторы могут управлять списками пользователей, моделей и стилей перевода, а также получать статистику жалоб по моделям и стилям перевода. Общее количество функций web-приложения составляет 22.
2. Рассмотрена логическая схема базы данных web-приложения, которая включает 13 таблиц. Таблицы хранят данные о пользователях, статьях, кон-

фигурациях и других.

3. Рассмотрена архитектура web-приложения. Использование RabbitMQ позволяет сервисам передавать сообщения между собой, Redis позволяет хранить данные с малым сроком жизни, а PostgreSQL – долговременные данные.

4 Реализация web-приложения

4.1 Обоснование выбора программной платформы

Для реализации web-приложения был выбран язык программирования Python [2] и фреймворк FastAPI [3]. FastAPI представляет собой web-фреймворк для создания API на языке Python. Он позволяет обрабатывать запросы асинхронно и поддерживает протокол WebSocket. Для сериализации, десериализации и валидации запросов использовалесь библиотека Pydantic [4].

Для долговременного хранения данных web-приложения была выбрана распространённая СУБД PostgreSQL [5], обладающая следующими преимуществами: бесплатность, расширяемость, большое сообщество, широкая поддержка среди инструментов разработки программного обеспечения.

Для создания моделей, соответствующих таблицам в реляционной базе данных, была выбрана библиотека SQLAlchemy [6]. Она предоставляет уровень абстракции над объектами базы данных, позволяя работать с ними как с объектами Python, а также предоставляет возможность создавать сложные запросы при помощи функций Python.

Для управления миграциями был выбран инструмент Alembic [7]. Данный инструмент позволяет отслеживать изменения в структуре базы данных, а также предоставляет возможность автоматической генерации миграций на основе изменений в моделях. Также Alembic предоставляет возможность отката к более ранней версии базы данных.

Для перевода текста используется сервис g4f [8]. Он выступает как посредник между web-приложением и публичными API различных провайдеров, обеспечивая работу web-приложения и упрощая его настройку. Сервис translation-consumer считывает задачи на перевод из очереди Kafka, отправляет запросы по указанному в переменных окружения адресу с необходимой полезной нагрузкой (текст, который нужно перевести, текст стиля перевода, название модели, название провайдера) и на основе полученных ответов создаёт объекты переведённых статей.

Для отправки электронной почты был выбран сервис Unisender [9], который предоставляет API для создания рассылок и отправки одиночных писем.

4.2 Реализация серверной части web-приложения

В соответствии с диаграммой вариантов использования функции, доступные пользователям, были реализованы в исходном коде. Исходный код web-приложения представлен в Приложении А.

4.2.1 Изменение учётной записи

Функция ”изменение учётной записи”(1) в исходном коде реализована функциями change_name, request_password_restoration_code и restore_password.

Функция change_name находится в модуле src.routers.users.views.py. Она принимает HTTP PATCH запрос по пути ”/users/user_id/name/”и изменяет имя пользователя на новое, полученное из тела запроса.

Функция request_password_restoration_code находится в модуле src.routers.auth.views.py. Она принимает HTTP POST запрос по пути ”/auth/restore-password/request/”с адресом электронной почты пользователя в параметрах строки запроса, создаёт код подтверждения смены пароля в базе данных и отправляет письмо по адресу электронной почты со ссылкой на страницу смены пароля.

Функция restore_password находится в модуле src.routers.auth.views.py. Она принимает HTTP POST запрос по пути ”/auth/restore-password/confirm/”с кодом подтверждения и новым паролем, проверяет существование полученного кода и изменяет хранимый в базе данных хэш пароля пользователя на хэш полученного нового пароля.

4.2.2 Просмотр открытых сессий

Функция ”просмотр открытых сессий”(2) в исходном коде реализована функцией get_sessions. Данная функция находится в модуле src.routers.sessions.views.py. Она принимает HTTP GET запрос по пути ”/sessions/”и возвращает список всех открытых сессий текущего пользователя из базы данных (сессий, у которых значение в столбце closed_at равняется NULL).

4.2.3 Завершение открытых сессий

Функция ”завершение открытых сессий”(3) в исходном коде реализована функцией close_sessions. Данная функция находится в модуле src.routers.sessions.views.py. Она принимает HTTP POST запрос по пути ”/sessions/close/”и закрывает все открытые сессии текущего пользователя (устанавливает в столбец closed_at текущее время сервера).

4.2.4 Изменение списка исходных статей

Функция ”изменение списка исходных статей”(4) в исходном коде реализована функциями upload_article, update_article, delete_article. Данные функции находятся в модуле src.routers.articles.views.py.

Функция upload_article находится в модуле src.routers.users.views.py. Она принимает HTTP POST запрос по пути ”/articles/”и добавляет в базу данных строку с информацией об исходной статье, десериализованной из тела запроса. Значение столбца user_id берётся из маркера доступа пользователя, передаваемого через Cookie.

Функция update_article находится в модуле src.routers.users.views.py. Она принимает HTTP PUT запрос по пути ”/articles/article_id/ проверяет принадлежность исходной статьи текущему пользователю по идентификатору статьи, полученному из пути запроса, и обновляет запись о статье согласно данным, десериа-

лизованным из тела запроса.

Функция delete_article находится в модуле src.routers.users.views.py. Она принимает HTTP DELETE запрос по пути ”/articles/article_id/ проверяет принадлежность исходной статьи текущему пользователю по идентификатору статьи, полученному из пути запроса, и удаляет исходную или переведённую статью по идентификатору.

4.2.5 Изменение списка переведённых статей

Функция ”изменение списка переведённых статей”(5) в исходном коде реализована функциями delete_article, рассмотренной выше, и create_translation.

Функция create_translation находится в модуле src.routers.translation.views.py. Она принимает HTTP POST запрос по пути ”/translation/”и отправляет в Kafka сообщение подписчику-переводчику о запуске перевода. Тело запроса должно содержать идентификатор статьи, массив идентификаторов конечных языков, на которые требуется выполнить перевод, а также идентификаторы стиля и модели перевода.

4.2.6 Изменение списка жалоб на переводы своих статей

Функция ”изменение списка жалоб на переводы своих статей”(6) в исходном коде реализована функциями create_report, update_report, update_report_status.

Функция create_report находится в модуле src.routers.reports.views.py. Она принимает HTTP POST запрос по пути ”/articles/article_id/report/ проверяет принадлежность переведённой статьи текущему пользователю по идентификатору статьи, полученному из пути запроса, и добавляет в базу данных строку с информацией о жалобе (текст и идентификатор причины жалобы из тела запроса).

Функция create_report находится в модуле src.routers.reports.views.py. Она принимает HTTP PUT запрос по пути ”/articles/article_id/report/ проверяет принадлежность переведённой статьи текущему пользователю и обновляет информацию о жалобе (текст и причина жалобы в теле запроса).

Функция create_report находится в модуле src.routers.reports.views.py. Она принимает HTTP PATCH запрос по пути ”/articles/article_id/report/status/ проверяет, имеет ли пользователь право устанавливать жалобе новый статус, и обновляет статус жалобы по идентификатору статьи.

4.2.7 Просмотр своих уведомлений

Функция ”просмотр своих уведомлений”(7) в исходном коде реализована функцией get_notifications_list. Данная функция находится в модуле src.routers.notifications.views.py. Она принимает HTTP GET запрос по пути ”/notifications/”и возвращает список непрочитанных (имеющих в столбце read_at значение NULL) уведомлений пользователя из базы данных.

4.2.8 Изменение списка комментариев к жалобам на переводы своих статей

Функция ”изменение списка комментариев к жалобам на переводы своих статей”(8) включает в себя ровно две функции: ”получение списка комментариев к жалобе”(9) и ”создание комментария”(10).

Функция ”получение списка комментариев к жалобе”(9) в исходном коде реализована функцией get_comments. Данная функция находится в модуле src.routers.reports.views.py. Она принимает HTTP GET запрос по пути ”/articles/article_id/report/comments/ проверяет, имеет ли право текущий пользователь получать список комментариев к этой жалобе по идентификатору статьи, полученному из пути запроса, и возвращает список комментариев к жалобе из базы данных.

Функция ”создание комментария”(10) в исходном коде реализована функцией create_comment. Данная функция находится в модуле src.routers.reports.views.py. Она принимает HTTP POST запрос по пути ”/articles/article_id/report/comments/”создаёт комментарий к жалобе по идентификатору статьи, к которой была оставлена жалоба.

4.2.9 Изменение списка настроек переводчика

Функция ”изменение списка настроек переводчика”(11) в исходном коде реализована функциями create_config, update_config и delete_config.

Функция create_config находится в модуле src.routers.config.views.py. Она принимает HTTP POST запрос по пути ”/configs/ проверяет, занято ли название конфигурации переводчика для данного пользователя, и создаёт новую строку в базе данных с полученным названием, идентификатором модели перевода и стиля перевода, а также массивом идентификаторов конечных языков.

Функция create_config находится в модуле src.routers.config.views.py. Она принимает HTTP PUT запрос по пути ”/configs/config_id/ проверяет принадлежность конфигурации текущему пользователю и занято ли новое название конфигурации переводчика для данного пользователя, и обновляет строку в базе данных согласно десериализованным из тела запроса данным.

Функция create_config находится в модуле src.routers.config.views.py. Она принимает HTTP DELETE запрос по пути ”/configs/config_id/ проверяет принадлежность конфигурации текущему пользователю и удаляет конфигурацию по её идентификатору.

4.2.10 Регистрация

Функция ”регистрация”(12) в исходном коде реализована функцией register. Данная функция находится в модуле src.routers.auth.views.py. Она принимает HTTP POST запрос по пути ”/auth/register/ проверяет, занят ли адрес электронной почты, и создаёт нового пользователя по имени, адресу электронной почты и паролю, по-

лученным из тела запроса. Значение столбца email_verified устанавливается в false, и пользователь должен дополнительно подтвердить свой адрес электронной почты.

4.2.11 Аутентификация

Функция ”аутентификация”(13) в исходном коде реализована функцией login. Данная функция находится в модуле src.routers.auth.views.py. Она принимает HTTP POST запрос по пути ”/auth/login/ проверяет существование пользователя с полученными из тела запроса адресом электронной почты и паролем и аутентифицирует пользователя: закрывает открытые сессии по IP-адресу и user_agent, полученными из заголовков запроса, создаёт новую сессию пользователя и возвращает пару маркеров для доступа к ресурсам и обновления маркеров.

4.2.12 Изменение списка открытых жалоб

Функция ”изменение списка открытых жалоб”(14) в исходном коде реализована функцией update_report_status. Данная функция находится в модуле src.routers.reports.views.py и была рассмотрена выше.

4.2.13 Создание комментариев для жалоб

Функция ”создание комментариев для жалоб”(15) в исходном коде реализована функцией create_comment. Данная функция находится в модуле src.routers.reports.views.py и была рассмотрена выше.

4.2.14 Просмотр статистики жалоб

Функция ”просмотр статистики жалоб”(16) в исходном коде реализована функциями get_models_stats и get_prompts_stats.

Функция get_models_stats находится в модуле src.routers.analytics.views.py. Она принимает HTTP GET запрос по пути ”/analytics/models-stats/”и возвращает статистику жалоб по всем моделям перевода в базе данных: сколько было подано жалоб на переводы по каждой модели и какие статусы у этих жалоб на данный момент.

Функция get_prompts_stats находится в модуле src.routers.analytics.views.py. Она принимает HTTP GET запрос по пути ”/analytics/prompts-stats/”и аналогична функции get_models_stats, но возвращает статистику по стилям перевода, а не моделям перевода.

4.2.15 Изменение списка стилей перевода

Функция ”изменение списка стилей перевода”(17) в исходном коде реализована функциями create_prompt, update_prompt и delete_prompt.

Функция create_prompt находится в модуле src.routers.prompts.views.py. Она

принимает HTTP POST запрос по пути ”/prompts/ проверяет, занято ли название стиля перевода существующей строкой в базе данных, и добавляет новый стиль перевода в базу данных по названию и тексту, полученным из тела запроса.

Функция update_prompt находится в модуле src.routers.prompts.views.py. Она принимает HTTP PUT запрос по пути ”/prompts/prompt_id/ проверяет, занято ли новое название стиля перевода, и обновляет строку в базе данных по идентификатору стиля перевода, полученному из пути запроса, согласно данным из тела запроса.

Функция delete_prompt находится в модуле src.routers.prompts.views.py. Она принимает HTTP DELETE запрос по пути ”/prompts/prompt_id/ проверяет существование стиля перевода по идентификатору из пути запроса и удаляет стиль по идентификатору.

4.2.16 Изменение списка моделей перевода

Функция ”изменение списка моделей перевода”(18) в исходном коде реализована функциями create_model, update_model и delete_model.

Функция create_model находится в модуле src.routers.models.views.py. Она принимает HTTP POST запрос по пути ”/models/ проверяет, занято ли название модели перевода, и добавляет в базу данных строку с отображаемым названием, внутренним названием и провайдером, полученным из тела запроса.

Функция update_model находится в модуле src.routers.models.views.py. Она принимает HTTP PUT запрос по пути ”/models/model_id/ проверяет, занято ли новое название модели перевода, и обновляет строку в базе данных по идентификатору модели, полученному из пути запроса, согласно данным, полученным из тела запроса.

Функция delete_model находится в модуле src.routers.models.views.py. Она принимает HTTP DELETE запрос по пути ”/models/model_id/”и удаляет модель перевода из базы данных по её идентификатору, полученному из тела запроса.

4.2.17 Изменение списка пользователей

Функция ”изменение списка пользователей”(19) в исходном коде реализована функциями create_user, update_user и delete_user.

Функция create_user находится в модуле src.routers.users.views.py. Она принимает HTTP POST запрос по пути ”/users/”и создаёт пользователя с заданным именем, адресом электронной почты, паролем и ролью. Эти данные десериализуются из тела запроса.

Функция update_user находится в модуле src.routers.users.views.py. Она принимает HTTP PUT запрос по пути ”/users/user_id/”и обновляет строку в базы данных по идентификатору пользователя, полученному из пути запроса, согласно данным, полученным из тела запроса.

Функция delete_user находится в модуле src.routers.users.views.py. Она принимает HTTP DELETE запрос по пути ”/users/user_id/”и удаляет пользователя по

идентификатору, полученному из пути запроса.

Функции ”создание пользователей”(20), ”создание модераторов”(21) и ”создание администраторов”(22) в исходном коде реализованы функцией src.routers.users.views.create_user, рассмотренной выше.

Для передачи данных от клиента серверу и обратно используется протокол HTTP и формат JSON. FastAPI автоматически проверяет тело запроса согласно указанной схеме, созданной при помощи Pydantic, что повысило читаемость кода.

4.3 Реализация базы данных

Согласно логической схеме базы данных, были созданы объекты базы данных. Модели SQLAlchemy объявлены в модуле src.database.models.py, представленном в Приложении А. Для изменения состояния базы данных использовался инструмент Alembic. Скрипт для создания базы данных и её объектов представлен в Приложении Б.

Для работы с базой данных в SQLAlchemy необходимо создать объект сессии. Предварительная настройка подключения представлена в листинге 7.

```
from sqlalchemy.ext.asyncio import \
    async_sessionmaker, \
    create_async_engine

engine = create_async_engine(Database.url)
Session = async_sessionmaker(engine)
```

Листинг 1 – Настройка подключения к базе данных

Затем необходимо создать экземпляр класса Session и работать с данным экземпляром. Класс Session предоставляет методы для добавления строк в базу данных (add), фиксации изменении в транзакции (commit), отката транзакции (rollback), закрытия сессии (close) и так далее. При помощи экземпляра данного класса можно выполнять операции с базой данных.

4.4 Реализация клиентской части web-приложения

Для реализации клиентской части web-приложения использовался фреймворк Vue [10] и библиотека компонентов Vuetify [11]. Фреймворк предоставляет широкие возможности по настройке приложения и повторному использованию кода, а библиотека предоставляет богатый выбор компонентов, которые можно использовать без тщательной настройки в виде, в котором они поставляются.

Для обеспечения навигации по сайту, выполненному по технологии одностраничного приложения, использовался встроенный инструмент VueRouter, позволяющий сопоставлять шаблонам пути к web-странице определённые компоненты, подставлять идентификаторы в качестве параметров к компонентам и использовать вложенные маршруты. Объявление сопоставления маршрутов компонентам представлено в листинге 14.

```
{ path: '/', redirect: '/landing' },
```

```
{ path: '/',
    component: BaseLayout,
    children: [
        { path: 'sessions', component: SessionsPage},
        { path: 'analytics', component: AnalyticsPage },
    ],
    props: true},
{ path: '/landing', component: LandingPage },
{ path: '/error', component: ErrorPage},
{ path: '/oauth/:provider/oauth-callback', component: OAuthCallback
  },
{ path:'/change-password', component: ConfirmPasswordChange },
{ path: '/confirm-email', component: ConfirmEmail }
```

Листинг 2 – Объявление сопоставления маршрутов компонентам

Для отрисовки текста статей, созданных в формате Markdown [12], использовалась библиотека marked. Она позволяет асинхронно отрисовывать текст в код HTML. Пример использования библиотеки marked представлен в листинге 20.

```
<template>
    <v-row>
        <div v-html="renderedMarkdown" class="markdown-renderer"></
           div>
    </v-row>
</template>
<script>
onMounted(async () => {
    const article_id = String(route.params.article_id)
    let response = await get_article(article_id)
    if (!response) {
        await router.push('/error')
    }
    Object.assign(article, response)
    renderedMarkdown.value = await marked(article.text);
    response = await fetch_data(`${Config.backend_address}/configs/`)
    if (response) {
        configs.value = response.data.list
    }
})
```

Листинг 3 – Использование библиотеки marked

Библиотека Vuetify предоставляет набор компонентов, ускоряющих создание клиентской части web-приложения. Пример использования компонентов библиотеки представлен в листинге 4.

```
<v-btn
    v-if="article.original_article_id === null"
><v-icon icon="mdi-earth"/></v-btn>
```

Листинг 4 – Использование компонентов из библиотеки Vuetify

Данная библиотека предоставляет и другие компоненты: таблицы, меню, раскрывающиеся списки и прочие.

4.5 Выводы

1. Web-приложение было реализовано с применением языка программирования Python и фреймворка FastAPI. Web-приложение реализует все заявленные функциональные возможности пользователей.
2. Для хранения данных использовалась СУБД PostgreSQL, для которой были созданы все необходимые объекты базы данных.
3. Клиентская часть web-приложения была реализована с применением фреймворка Vue и библиотеки компонентов Vuetify.

5 Тестирование web-приложения

5.1 Функциональное тестирование

Для тестирования работоспособности web-приложения необходимо добавить объекты пользователя и модератора в таблицу Users (роли “user” и “moderator” соответственно; имя, пароль и адрес электронной почты произвольные). Объект администратора, а также языки, модели, стили перевода и причины для жалоб добавляются в базу автоматически при развёртывании web-приложения в Docker Compose. Также автоматически создаётся база данных, и в ней создаются все необходимые объекты.

Для проверки функций приложения рекомендуется использовать инструмент OpenAPI. Данный инструмент генерирует документацию на основе исходного кода web-приложения. Фреймворк FastAPI включает данный инструмент, и страница документации по умолчанию доступна по IP-адресу сервера, на котором развёрнуто приложение, и пути запроса /api/docs. Внешний вид данной страницы представлен в Приложении В. Для отправки запроса необходимо кликнуть по нужному элементу списка, нажать на кнопку “Try it out” ввести необходимые данные (тело запроса и его параметры) и нажать на кнопку “Execute”. Описание тестирования функций web-приложения представлено в таблице 5.1.

Таблица 5.1 – Описание тестирования функций web-приложения

Функция web-приложения	Описание тестирования	Итог тестирования функции
1 Изменение учётной записи	Аутентифицироваться в качестве пользователя, получить идентификатор своего пользователя при помощи GET запроса по адресу /api/users/me/, отправить POST запрос по пути /api/users/идентификатор своего пользователя/name/, указав в теле запроса желаемое имя в параметре name (формат тела запроса – JSON). Сервер должен вернуть объект обновлённого пользователя в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
2 Просмотр открытых сессий	Аутентифицироваться в качестве пользователя, отправить GET запрос на адрес /api/sessions/. Сервер должен вернуть список сессий в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено

Продолжение таблицы 5.1

3 Завершение открытых сессий	Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/sessions/close/. Сервер должен вернуть сообщение об успешном закрытии всех сессий	Работоспособность функции протестирована, ошибок не обнаружено
4 Изменение списка исходных статей	Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/articles/, указав в теле запроса заголовок (title), текст (text) и идентификатор языка (language_id) загружаемой статьи в формате JSON. Сервер должен вернуть объект статьи в формате JSON. Получить список языков в формате JSON можно, отправив GET запрос на адрес /api/languages/. Сервер должен вернуть список в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
5 Изменение списка переведённых статей	Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/translation/, указав в теле запроса идентификатор статьи, которую нужно перевести (article_id), список идентификаторов языков, на которые нужно перевести статью (target_language_ids), идентификатор стиля перевода (prompt_id) и идентификатор модели перевода (model_id). Сервер должен вернуть сообщение о запуске перевода, через некоторое время, зависящее от объёма статьи, в таблице Notifications должна появиться запись об успешном или неуспешном переводе статьи. Списки моделей и стилей перевода можно получить по GET запросам на адреса /api/models/ и /api/prompts/ соответственно	Работоспособность функции протестирована, ошибок не обнаружено

Продолжение таблицы 5.1

6 Изменение списка жалоб на переводы своих статей	Аутентифицироваться в качестве пользователя, отправить запрос на адрес /api/articles/идентификатор статьи/report/, в теле запроса указать текст жалобы (text) и идентификатор причины жалобы (reason_id). Сервер должен вернуть объект жалобы в формате JSON. Список доступных причин жалоб можно получить при помощи GET запроса на адрес /api/report-reasons/	Работоспособность функции протестирована, ошибок не обнаружено
7 Просмотр своих уведомлений	Аутентифицироваться в качестве пользователя, отправить GET запрос на адрес /api/notifications/. Сервер должен вернуть список непрочитанных уведомлений в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
8 Изменение списка комментариев к жалобам на переводы своих статей	Протестировать функцию 10 Создание комментария, затем функцию 9 Получение списка комментариев к жалобе	Работоспособность функции протестирована, ошибок не обнаружено
9 Получение списка комментариев к жалобе	Аутентифицироваться в качестве пользователя, отправить GET запрос на адрес /api/articles/ идентификатор переведённой статьи, для жалобы на которую требуется получить список комментариев /report/comments/. Сервер должен вернуть список комментариев в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
10 Создание комментария	Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/articles/ идентификатор переведённой статьи, для жалобы на которую требуется создать комментарий /report/comments/, в запросе указать текст комментария (text). Сервер должен вернуть объект комментария в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено

Продолжение таблицы 5.1

11 Изменение списка настроек переводчика	Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/configs/, в запросе указать название конфигурации (name), идентификатор стиля перевода (prompt_id), идентификатор модели перевода (model_id) и список конечных языков (language_ids). Сервер должен вернуть объект конфигурации в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
12 Регистрация	Отправить POST запрос на адрес /api/auth/register/, указав в теле запроса имя пользователя (name), адрес электронной почты (email) и пароль (password). Сервер должен вернуть сообщение об успешной регистрации	Работоспособность функции протестирована, ошибок не обнаружено
13 Аутентификация	Отправить POST запрос на адрес /api/auth/login/, указав в теле запроса адрес электронной почты (email) и пароль (password). Сервер должен вернуть сообщение об успешной аутентификации	Работоспособность функции протестирована, ошибок не обнаружено
14 Изменение списка открытых жалоб	Аутентифицироваться в качестве модератора, отправить на адрес /api/articles/ идентификатор статьи, жалобу на которую нужно изменить /report/status/ POST запрос, указав в параметрах запроса новый статус жалобы (Отклонена или Удовлетворена). Сервер должен вернуть объект жалобы в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
15 Создание комментариев для жалоб	Аутентифицироваться в качестве модератора, отправить на адрес /api/articles/ идентификатор статьи, для жалобы на которую нужно создать комментарий /report/comments/ POST запрос, указав в теле запроса текст комментария (text). Сервер должен вернуть объект комментария в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
16 Просмотр статистики жалоб	Аутентифицироваться в качестве администратора, отправить GET запрос на адрес /api/analytics/models-stats/. Сервер должен вернуть данные по жалобам для каждой модели перевода в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено

Продолжение таблицы 5.1

17 Изменение списка стилей перевода	Аутентифицироваться в качестве администратора, отправить POST запрос на адрес /api/prompts/, в теле запроса указать название (title) и текст (text) стиля перевода Сервер должен вернуть объект стиля перевода в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
18 Изменение списка моделей перевода	Аутентифицироваться в качестве администратора, отправить POST запрос на адрес /api/models/, в теле запроса указать отображаемое название (show_name), название (name) и провайдер (provider) модели перевода Сервер должен вернуть объект модели перевода в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
19 Изменение списка пользователей	Аутентифицироваться в качестве администратора, отправить POST запрос на адрес /api/users/, в теле запроса указать имя (name), адрес электронной почты (email), флаг, указывающий, подтверждена ли почта (email_verified), роль (role) и пароль (password) пользователя. Сервер должен вернуть объект созданного пользователя в формате JSON	Работоспособность функции протестирована, ошибок не обнаружено
20 Создание пользователей	Аналогично тестированию функции 19 Изменение списка пользователей, но роль в теле запроса должна быть “Пользователь”	Работоспособность функции протестирована, ошибок не обнаружено
21 Создание модераторов	Аналогично тестированию функции 19 Изменение списка пользователей, но роль в теле запроса должна быть “Модератор”	Работоспособность функции протестирована, ошибок не обнаружено
22 Создание администраторов	Аналогично тестированию функции 19 Изменение списка пользователей, но роль в теле запроса должна быть “Администратор”	Работоспособность функции протестирована, ошибок не обнаружено

Таким образом, были протестированы все ключевые функции web-приложения, ошибок не обнаружено.

5.2 Нагрузочное тестирование

Нагрузочное тестирование является критическим компонентом обеспечения надежности и производительности веб-приложений. Его целью является выявление определение максимальной пропускной способности приложения, идентификация предельных характеристик и количественная оценка производительности приложения в различных условиях.

В силу того, что web-приложение не выполняет задач, требующих большого времени центрального процессора, основную часть времени обработки запроса занимает работа с базой данных. Для проверки поведения приложения под нагрузкой был разработан модуль tests.hot_load.py, представленный в Приложении А. Данный модуль реализует класс HotLoad, предназначенный для отправки большого количества запросов на протяжении заданного времени. Для этого он использует класс Pool стандартного пакета multiprocessing. Использование нескольких процессов позволяет избежать ошибок отправки запросов из одного потока, при которых запросы не отправляются полностью.

Функции, предназначенные для запуска процессов в данном классе, представлена в листинге 19.

```
    def run_process(self, process_number: int, *args) -> int:
        worker_start_id = process_number * self.workers_number
        loop = asyncio.get_event_loop()
        result = loop.run_until_complete(self.run_workers(
            worker_start_id))
        return result

    async def run(self) -> float:
        if self.on_startup_callable:
            self.headers = await self.on_startup_callable()

        with multiprocessing.Pool(processes=self.processes_number) as
            pool:
            results = pool.map(self.run_process, range(self.
                processes_number))
        mean_rps = sum(results) / self.duration.total_seconds()

        if self.on_teardown_callable:
            await self.on_teardown_callable()

        return mean_rps
```

Листинг 5 – Функции запуска процессов класса HotLoad

Для запуска теста необходимо выполнить команду “docker exec docker-api-1 bash -c “python tests/test_six_hot_loads.py””. Данный тест выполняет повторяющиеся GET и POST запросы к серверу при помощи шести дочерних процессов на протяжении 30 секунд. По истечении заданного времени в терминал будет выведено среднее количество выполненных запросов в секунду.

Тестирование выявило некоторые ошибки в исходном коде. В частности, од-

на сессия базы данных использовалась в разных обработчиках конкурентно, что приводило к ошибкам. Ошибка была решена использованием примитива синхронизации Semaphore из стандартного пакета asyncio. Тестирование показало высокую пропускную способность приложения: порядка 80 запросов в секунду для запросов, получающих данные из базы данных и добавляющих данные в неё.

5.3 Выводы

1. Все функциональные возможности пользователей были протестированы, обнаруженные ошибки были исправлены.
2. Web-приложение было протестировано в условиях поступления большого количества запросов, по результатам которого показало высокую пропускную способность и устойчивость к нагрузкам.

6 Руководство программиста

6.1 Настройка окружения

Приложение разворачивалось на системе Ubuntu Server 24.04. Для корректной работы необходимо выполнить следующие шаги:

- включить Uncomplicated Firewall при помощи команды “sudo ufw enable”;
- добавить перенаправление портов для доступа к web-приложению при помощи команд “sudo ufw allow 80” и “sudo ufw allow 443”;
- опционально включить доступ по SSH при помощи команды “sudo ufw allow ssh” для доступа с удалённой машины;
- получить IP-адрес сервера при помощи команды “ip a”;
- занести полученный IP-адрес в файл hosts в формате “192.168.122.233 ugabuntu.com”;
- создать в домашнему каталоге серверного пользователя папку проекта web-приложения, в которой будут находиться необходимые файлы, и перейти в неё при помощи команды “mkdir gptranslate && cd gptranslate”;
- создать в папке все необходимые файлы, представленные в Приложении А и Приложении Г.

Для развёртывания web-приложения применяется инструмент Docker Compose. Перед развёртыванием web-приложения необходимо убедиться, что в системе установлены Docker Engine и Docker Compose при помощи команд docker version и docker compose version. В случае, если любая из указанных технологий не установлена, её необходимо установить согласно подходящей инструкции на официальном сайте, например, [13] для Docker Engine и [14] для Docker Compose.

Для корректного функционирования web-приложения необходимо создать сеть Docker при помощи команды “docker network create a”. Данная сеть объединяет контейнеры в рамках Docker Compose и позволяет им коммуницировать между собой. Также данная сеть позволяет подключать к web-приложению внешние сервисы, развёрнутые на локальной машине в Docker, но не входящие в один проект Docker Compose с web-приложением.

Для корректной работы web-приложения ему необходим доступ к внешнему сервису g4f. Он может находиться в любом удобном месте: на локальной машине или на удалённом сервере. Для большего удобства можно развернуть его в Docker и добавить в ранее созданную сеть. Для этого нужно скачать базовый образ при помощи команды “docker pull hlohaus789/g4f:0.3.9.7”, развернуть его при помощи команды “docker run –detach –name g4f hlohaus789/g4f:0.3.9.7”, добавить созданный контейнер в сеть при помощи команды “docker network connect a g4f”.

6.2 Развёртывание приложения

В папке web-приложения необходимо создать файл .env, в котором нужно указать необходимые значения переменных окружения, используемых web-приложением, таких как ключ доступа Unisender, логин и пароль для доступа к

базе данных и так далее. Примеры объявления переменных окружения находится в файле .example.env. За адрес сервиса g4f отвечает переменная G4F_ADDRESS. Ей необходимо присвоить адрес данного сервиса в формате “http://address:port”. В случае, если данный сервис был развёрнут на локальной машине в Docker согласно вышеуказанной инструкции, его адрес будет равен “http://g4f:1337”.

Далее в корневой папке web-приложения необходимо последовательно выполнить команды “docker build -t diploma-base -f contrib/docker/base/Dockerfile .” и “docker compose –env-file=.env -f contrib/docker/docker-compose.prod.yaml up -d –build”. Эти команды создадут новую сеть Docker, соберут базовый образ для контейнеров из исходного кода и запустят все необходимые контейнеры соответственно. Проверить доступность сервиса g4f можно при помощи команды “docker exec -t docker-api-1 bash -c ”/app/contrib/docker/wait-for-it.sh \”g4f:1337\t 30 – echo \”Сервис доступен\”.

В папке contrib/persistent_data находятся .json файлы с данными, которыми будет заполнена база данных по умолчанию:

- languages.json хранит информацию о доступных для перевода языках в формате словаря, чьими ключами являются названия языков, а значениями
- их трёхбуквенные коды ISO 639-3:2007;
- models.json хранит массив массивов, хранящих отображаемое название модели и внутренние названия модели и провайдера, используемые для запросов к сервису g4f;
- prompts.json хранит массив массивов, хранящих название стиля перевода и текст стиля перевода;
- report-reasons.json хранит массив словарей с идентификатором, названием и позицией при сортировке.

При каждом запуске контейнера api будет производиться проверка на наличие данных, которых нет в базе данных, и отсутствующие строки будут добавлены автоматически.

Также при запуске контейнера api автоматически создаётся администратор с адресом электронной почты admin@d.com и паролем string и производится обновление структуры базы данных в соответствии с файлами миграций, находящихся в папке src/database/alembic/versions.

6.3 Проверка работоспособности приложения

После развёртывания web-приложения по адресу https://localhost будет доступна web-страница web-приложения. Также приложение должно быть доступно с других компьютеров в локальной сети по IP-адресу хоста. Шаги по проверке работоспособности развёрнутого web-приложения описаны в разделе 5.

6.4 Выводы

– Было создано руководство, позволяющее развернуть и протестировать работоспособность web-приложения в системе Ubuntu Server 24.04.
– Руководство также описывает локальное развёртывание сервиса g4f, используемого web-приложением.

7 Технико-экономическое обоснование проекта

7.1 Общая характеристика разрабатываемого программного средства

Основной целью экономического раздела является экономическое обоснование целесообразности разработки web-приложения, представленного в дипломном проекте. В данном разделе проводится расчет затрат на всех стадиях разработки, а также анализ экономического эффекта в связи с использованием данного веб-приложения.

Разработанное web-приложение позволяет пользователям переводить значительные объёмы текста с одного языка на множество других языков.

Во время разработки дипломного проекта использовалась технология FastAPI для написания серверной части приложения и библиотека Vue.js для написания клиентской части приложения. Данное web-приложение разработано для последующего использования в коммерческих целях.

7.2 Методика обоснования цены

В современных рыночных экономических условиях web-приложение выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид web-приложения, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки web-приложения являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования web-приложения. Выбор эффективных проектов web-приложения связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации приложения, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, финансовых ресурсов, получаемой за счет:

- снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
- снижения расходов на материалы;
- ускорение ввода в эксплуатацию новых систем;
- улучшения показателей основной деятельности в результате использования веб-приложения.

Стоимостная оценка веб-приложения у разработчиков предполагает определение затрат, что включает следующие статьи:

- заработная плата исполнителей – основная и дополнительная;
- отчисления в фонд социальной защиты населения;
- отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
- расходы на оплату машинного времени;
- накладные расходы;

– прочие прямые затраты.

Для расчёта стоимости разработки web-приложения необходимо установить определённые параметры, представленные в таблице 7.1.

Таблица 7.1 – Параметры, применяемые при расчёте стоимости разработки

Параметр	Значение
Норматив ФСЗН+БГС	0,346
Норматив доп. ЗП	0,15
Норматив прочих затрат	0
Норматив накладных расходов	0,5
Норматив расходов на реализацию	0,1
Ставка НДС	20%
Прочие прямые расходы (стоимость подписки использования GPT, аренда сервера для разработки и тестирования)	149,1 руб.
Повышающий коэффициент ЗП	1

На основании затрат рассчитывается себестоимость и отпускная цена конечного web-приложения.

7.2.1 Стоимость разработки

Стоимость разработки напрямую зависит от заработной платы специалистов и их трудозатрат. Для определения величины основной заработной платы было проведено исследование заработных плат для специалистов в сфере веб-разработки. Источником данных служили открытые веб-порталы, различные форумы и общий средний уровень заработке в сфере информационных технологий. Было установлено, что средняя месячная заработная плата дизайнера составляет 20 рублей в час, бизнес-аналитика – 20.83 рубля в час, технического лидера – 100 рублей в час, junior бэкенд/фронтенд разработчика и тестировщика – 10 рублей в час, middle бэкенд разработчика – 30 рублей в час, middle фронтенд разработчика – 28 рублей в час, middle тестировщика – 24 рубля в час.

Проект разрабатывался командой из бизнес-аналитика, технического лидера, дизайнера, а также junior и middle фронтенд и бекэнд разработчиков и тестировщиков на протяжении двух месяцев. Трудозатраты каждого работника представлены в таблице 7.2.

Таблица 7.2 – Трудозатраты и ставки оплаты работников

Базовая ставка в час, руб	Специалист	Трудозатраты, ч	Ставка в час, руб
20	Дизайнер	120	20
20,83	Бизнес-аналитик	24	20,83
100	Технический лидер	16	100

Продолжение таблицы 7.2

10	junior бэкенд разработчик	140	10
30	middle бэкенд разработчик	140	30
10	junior фронтенд разработчик	124	10
28	middle фронтенд разработчик	124	28
10	junior тестировщик	40	10
24	middle тестировщик	40	24

Затраты на заработную плату каждому работнику в зависимости от ставки и трудозатрат определяются по формуле 7.1.

$$С_р = С \cdot Т \cdot ((1 + Н_{доп.\ ЗП}) \cdot (1 + Н_{ФСЗН}) + Н_п + Н_н) \quad (7.1)$$

где $С_р$ – стоимость разработки;

С – ставка работника в час;

Т – трудозатраты работника в часах;

$ЗП_{осн}$ – основная зарплата;

$ЗП_{доп}$ – дополнительная зарплата;

$Н_{ФСЗН}$ – норматив отчислений в ФСЗН;

$Н_{доп.\ ЗП}$ – норматив дополнительной зарплаты;

$Н_п$ – норматив прочих затрат;

$Н_н$ – норматив накладных расходов.

Таким образом было определено, что стоимость разработки составляет 33267.57 рублей.

7.2.2 Цена продажи

При расчёте цены продажи было решено использовать желаемую маржинальность 20%. Таким образом, определить цену с НДС можно по формуле 7.2.

$$Ц_{НДС} = С_р(1 + Р)(1 + С_{НДС}) \quad (7.2)$$

где $Ц_{НДС}$ – цена продажи, включая НДС;

$С_{НДС}$ – ставка НДС;

Р – рентабельность.

Заключение

Таким образом, было создано web-приложение «GPTranslate» для перевода текста ограниченного объёма с иностранного языка с применением сервиса «g4f». Web-приложение обладает следующими характеристиками:

- использует четыре роли пользователей: Гость, Пользователь, Модератор, Администратор. Каждая роль может выполнять свои функции;
- реализует 22 ключевые функции;
- использует 13 таблиц в базе данных;
- реализовано согласно монолитной архитектуре с применением вспомогательных компонентов, таких как подписчики RabbitMQ;
- объём исходного кода порядка 8000 строк;
- протестировано с применением ручного и нагрузочного тестирования. Unit-тесты не создавались, покрытие тестами отсутствует.

Разработанное web-приложение представляет собой комплексный инструмент, который эффективно решает проблему быстрого и качественного перевода текстов с использованием существующих сервисов, предоставляющих доступ к нейронным сетям. Монолитная архитектура приложения, основанная на языке программирования Python и фреймворке FastAPI, обеспечивает высокую производительность и удобство использования.

Проведённое тестирование подтвердило корректность работы программного продукта и его соответствие заявленному функционалу. Подготовленная техническая документация упрощает развёртывание web-приложения. Полный исходный код web-приложение находится по адресу https://github.com/XoJIoDuJIHuK/diploma.

Список использованных источников

1. PostgreSQL Documentation: 15: Chapter 8. Data Types [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.postgresql.org/docs/15/datatype.html;
2. Our Documentation | Python.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.python.org;
3. FastAPI [Электронный ресурс]. – Электронные данные. – Режим доступа: https://fastapi.tiangolo.com;
4. Welcome to Pydantic – Pydantic [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.pydantic.dev/latest/;
5. PostgreSQL: Feature Matrix [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.postgresql.org/about/featurematrix/;
6. SQLAlchemy - The Database Toolkit for Python [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.sqlalchemy.org;
7. Welcome to Alembic's documentation! — Alembic 1.14.0 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://alembic.sqlalchemy.org;
8. xtekky/gpt4free: The official gpt4free repository — GitHub [Электронный ресурс]. – Электронные данные. – Режим доступа: https://github.com/xtekky/gpt4free;
9. Документация API для email-рассылок в Unisender [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.unisender.com/ru/support/api/common/bulk-email/;
10. Introduction | Vue.js [Электронный ресурс]. – Электронные данные. – Режим доступа: https://vuejs.org/guide/introduction.html;
11. Vuetify — A Vue Component Framework [Электронный ресурс]. – Электронные данные. – :https://vuetifyjs.com/en/;
12. Markdown Cheat Sheet [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.markdownguide.org/cheat-sheet/;
13. Install | Docker Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/engine/install/;
14. Install | Docker Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/compose/install/.

Приложение А

Исходный код наиболее значимых частей web-приложения

```
//src.routers.analytics.views.py
@router.get(
    '/models-stats/'
)
async def get_models_stats(
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin])),
        db_session: AsyncSession = Depends(get_session)
):
    return await AnalyticsRepo.get_models_stats(db_session)


@router.get(
    '/prompts-stats/'
)
async def get_prompts_stats(
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin])),
        db_session: AsyncSession = Depends(get_session)
):
    return await AnalyticsRepo.get_prompts_stats(db_session)




//src.routers.articles.views.py
@router.post(
    '/',
    response_model=DataResponse.single_by_key(
        'article',
        ArticleOutScheme
    ),
    responses=get_responses(400, 401, 403, 500)
)
async def upload_article(
        article_data: UploadArticleScheme,
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user])),
        db_session: AsyncSession = Depends(get_session)
):
    article = await ArticleRepo.create(
        article_data=CreateArticleScheme(
            title=article_data.title,
            text=article_data.text,
            language_id=article_data.language_id,
            user_id=user_info.id
        ),
        db_session=db_session
    )
    return DataResponse(
        data={
```

```
            'article': ArticleOutScheme.model_validate(article)
        }
    )


@router.put(
    '/{article_id}/',
    response_model=DataResponse.single_by_key(
        'article',
        ArticleOutScheme
    ),
    responses=get_responses(400, 401, 403, 404, 500)
)
async def update_article(
        new_article_data: EditArticleScheme,
        article_id: uuid.UUID = Path(),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user])),
        db_session: AsyncSession = Depends(get_session),
):
    article = await ArticleRepo.get_by_id(article_id, db_session)
    if (
            not article
            or article.user_id != user_info.id
            or article.original_article_id is not None
    ):
        raise article_not_found_error
    if new_article_data.title is not None:
        article.title = new_article_data.title
    if new_article_data.text is not None:
        article.text = new_article_data.text
    db_session.add(article)
    await db_session.commit()
    await db_session.refresh(article)
    return DataResponse(
        data={
            'article': ArticleOutScheme.model_validate(article)
        }
    )


@router.delete(
    '/{article_id}/',
    response_model=DataResponse.single_by_key(
        'article',
        ArticleOutScheme
    ),
    responses=get_responses(400, 401, 403, 404, 500)
)
async def delete_article(
        article_id: uuid.UUID = Path(),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user])),
```

```
        db_session: AsyncSession = Depends(get_session),
):
    article = await ArticleRepo.get_by_id(article_id, db_session)
    if not article or article.user_id != user_info.id:
        raise article_not_found_error
    article = await ArticleRepo.delete(
        article=article,
        db_session=db_session
    )
    return DataResponse(
        data={
            'article': ArticleOutScheme.model_validate(article)
        }
    )




//src.routers.auth.views.py
@router.post(
    '/login/',
    responses=get_responses(404)
)
async def login(
        login_data: LoginScheme,
        request: Request,
        db_session: AsyncSession = Depends(get_session)
):
    user = await UserRepo.get_by_email(
        email=login_data.email,
        db_session=db_session
    )
    if (
        not user or
        user.password_hash != get_password_hash(login_data.password)
    ):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail      ='          '
        )
    if not user.email_verified:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail      ='              '
        )
    if AppConfig.close_sessions_on_same_device_login:
        await SessionRepo.close_all(
            user_id=user.id,
            ip=request.headers.get('X-Forwarded-For'),
            user_agent=get_user_agent(request),
            db_session=db_session
```

```
        )
    await db_session.refresh(user)
    tokens = await AuthHandler.login(
        user=user,
        request=request,
        db_session=db_session
    )
    response = JSONResponse({'detail':          ''})
    return get_authenticated_response(response, tokens)


@router.post(
    '/register/',
    response_model=BaseResponse,
    responses=get_responses(409)
)
async def register(
        registration_data: RegistrationScheme,
        db_session: AsyncSession = Depends(get_session)
):
    if await UserRepo.name_is_taken(
        name=registration_data.name,
        db_session=db_session
    ):
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail ='    '
        )
    user = await UserRepo.create(
        user_data=CreateUserScheme(
            name=registration_data.name,
            email=registration_data.email,
            email_verified=False,
            password=registration_data.password,
            role=Role.user
        ),
        db_session=db_session
    )
    await send_email_confirmation_message(
        user=user,
        email=registration_data.email,
        db_session=db_session
    )
    return BaseResponse(message      ='    .          ')


@router.post(
    '/restore-password/request/',
    response_model=BaseResponse,
    responses=get_responses(404)
)
```

```
async def request_password_restoration_code(
        email: EmailStr,
        db_session: AsyncSession = Depends(get_session)
):
    user = await UserRepo.get_by_email(
        email=email,
        db_session=db_session
    )
    if not user:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail      ='                '
        )
    confirmation_code = await ConfirmationCodeRepo.create(
        user_id=user.id,
        reason=ConfirmationType.password_reset,
        db_session=db_session
    )
    producer = KafkaProducer(
        bootstrap_servers=KafkaConfig.address,
        topic=KafkaConfig.mail_topic
    )
    kafka_message = SendEmailScheme(
        to_address=email,
        from_address=UnisenderConfig.from_address,
        from_name=UnisenderConfig.from_name,
        subject=UnisenderConfig.password_recovery_subject,
        template_id=UnisenderConfig.password_recovery_template_id,
        params={
            'link': f'{FrontConfig.address}'
                    f'{FrontConfig.change_password_endpoint}'
                    f'?code={confirmation_code.code}'
        }
    )
    await producer.send_message(kafka_message.model_dump(mode='json')
       )
    return BaseResponse(message     ='                ')


@router.patch(
    '/restore-password/confirm/',
    response_model=BaseResponse,
    responses=get_responses(400, 404)
)
async def restore_password(
        request_data: ResetPasswordScheme,
        db_session: AsyncSession = Depends(get_session)
):
    confirmation_code = await ConfirmationCodeRepo.get(
        value=request_data.code,
        reason=ConfirmationType.password_reset,
```

```
        db_session=db_session
    )
    if not confirmation_code:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail ='                    '
        )
    new_password_hash = get_password_hash(request_data.new_password)
    await UserRepo.update_password_hash(
        user_id=confirmation_code.user_id,
        new_password_hash=new_password_hash,
        db_session=db_session
    )
    await ConfirmationCodeRepo.mark_as_used(
        confirmation_code=confirmation_code,
        db_session=db_session
    )
    return BaseResponse(message    ='          ')



//src.routers.config.views.py
@router.post(
    '/',
    response_model=DataResponse.single_by_key(
        'config',
        ConfigOutScheme
    ),
    responses=get_responses(400, 401, 409)
)
async def create_config(
        request: Request,
        config_data: CreateConfigScheme,
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user]))
):
    config = await ConfigRepo.create(
        config_data=config_data,
        user_id=user_info.id,
        db_session=db_session
    )
    return DataResponse(
        data={
            'config': ConfigOutScheme.model_validate(config)
        }
    )


@router.put(
    '/{config_id}/',
```

```
    response_model=DataResponse.single_by_key(
        'config',
        ConfigOutScheme
    ),
    responses=get_responses(400, 401, 404, 409)
)
async def update_config(
        config_data: EditConfigScheme,
        config: TranslationConfig = Depends(get_config),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user]))
):
    config = await ConfigRepo.update(
        config=config,
        new_data=config_data,
        db_session=db_session
    )
    return DataResponse(
        data={
            'config': ConfigOutScheme.model_validate(config)
        }
    )


@router.delete(
    '/{config_id}/',
    response_model=BaseResponse,
    responses=get_responses(400, 401, 404, 409)
)
async def delete_config(
        request: Request,
        config: TranslationConfig = Depends(get_config),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user]))
):
    logger.info(f'Worker {request.headers.get('X-Worker-ID', 'unknown
       ')} is trying to delete config {config.name[-1]}')
    config_name = config.name
    await ConfigRepo.delete(
        config=config,
        db_session=db_session
    )
    return BaseResponse(message=f    ' {config_name}     ')




//src.routers.models.views.py
@router.post(
    '/',
    response_model=DataResponse.single_by_key(
```

```
        'model',
        ModelOutScheme
    ),
    responses=get_responses(400, 401, 403, 409)
)
async def create_model(
        model_data: ModelCreateScheme,
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin]))
):
    await check_model_conflicts(
        model_data=model_data,
        existing_model_id=None,
        db_session=db_session
    )
    model = await ModelRepo.create(
        model_data=model_data,
        db_session=db_session
    )
    return DataResponse(
        data={
            'model': ModelOutScheme.model_validate(model)
        }
    )


@router.put(
    '/{model_id}/',
    response_model=DataResponse.single_by_key(
        'model',
        ModelOutScheme
    ),
    responses=get_responses(400, 401, 403, 404, 409)
)
async def update_model(
        model_data: ModelUpdateScheme,
        model_id: int = Path(),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin]))
):
    model = await ModelRepo.get_by_id(
        model_id=model_id,
        db_session=db_session
    )
    if not model:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail   ='        '
        )
    await check_model_conflicts(
        model_data=model_data,
```

```
        existing_model_id=model_id,
        db_session=db_session
    )
    model = await ModelRepo.update(
        model=model,
        new_model_data=model_data,
        db_session=db_session
    )
    return DataResponse(
        data={
            'model': ModelOutScheme.model_validate(model)
        }
    )


@router.delete(
    '/{model_id}/',
    response_model=BaseResponse,
    responses=get_responses(400, 401, 403, 404)
)
async def delete_model(
        model_id: int = Path(),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin]))
):
    result = await ModelRepo.delete(
        model_id=model_id,
        db_session=db_session
    )
    return BaseResponse(message=result)




//src.routers.oauth.views.py
@router.get(
    '/login/'
)
async def redirect_to_provider(
        request: Request,
        provider: OAuthProvider,
):
    provider_authorize = get_oauth_provider(
        provider=provider,
        storage=RedisHandler()
    )
    new_session_data = {
        OAuthConfig.session_data_property: {
            'ip': request.headers.get('X-Forwarded-For'),
        }
    }
```

```
    request.session.update(new_session_data)
    authorization_url = await provider_authorize.get_auth_url()
    return RedirectResponse(authorization_url)


@router.get(
    '/{provider}/callback',
    summary='Validates auth code from provider and returns user\'s
       tokens',
    response_model=None
)
async def callback(
    request: Request,
    provider: OAuthProvider = Path(),
    db_session: AsyncSession = Depends(get_session),
):
    oauth_login_data = request.session.get(
        OAuthConfig.session_data_property
    )
    if not oauth_login_data:
        error_message = (
            f'          : {request.session},          '
            f' \'{OAuthConfig.session_data_property}\''
        )
        logger.error(error_message)
        raise Exception(error_message)


    provider_authorize = get_oauth_provider(
        provider=provider,
        storage=RedisHandler()
    )
    auth_token = await provider_authorize.callback(
        request=request
    )


    user_data = await provider_authorize.get_user_info(auth_token)
    logger.error(user_data)
    user_id = user_data.id
    provider_user_id = (str(user_id) if user_id else None)


    if email := user_data.email:
        user = await UserRepo.get_by_email(
            email=email,
            db_session=db_session
        )
        if not user:
            user = await UserRepo.register_for_oauth(
                role=Role.user,
```

```
                db_session=db_session,
                email=email,
                name=user_data.name,
                oauth_provider=provider,
                provider_id=provider_user_id,
            )
    else:
        user = await UserRepo.get_by_oauth_data(
            provider=provider,
            provider_id=provider_user_id,
            db_session=db_session
        )
        if not user:
            user = await UserRepo.register_for_oauth(
                email=None,
                name=user_data.name,
                role=Role.user,
                db_session=db_session,
                oauth_provider=provider,
                provider_id=provider_user_id,
            )
    db_session.add(user)
    await db_session.commit()
    await db_session.refresh(user)
    tokens = await AuthHandler.login(
        user=user,
        request=request,
        db_session=db_session
    )
    response = RedirectResponse(f'/')
    return get_authenticated_response(response, tokens)




//src.routers.prompts.views.py
@router.post(
    '/',
    response_model=DataResponse.single_by_key(
        'prompt',
        PromptOutScheme
    )
)
async def create_prompt(
        prompt_data: CreatePromptScheme,
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin]))
):
    prompt = await PromptRepo.create(
        prompt_data=prompt_data,
        db_session=db_session
```

```
    )
    return DataResponse(
        data={
            'prompt': PromptOutScheme.model_validate(prompt)
        }
    )


@router.put(
    '/{prompt_id}/',
    response_model=DataResponse.single_by_key(
        'prompt',
        PromptOutScheme
    )
)
async def update_prompt(
        prompt_data: EditPromptScheme,
        prompt: StylePrompt = Depends(get_prompt),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin]))
):
    prompt = await PromptRepo.update(
        prompt=prompt,
        prompt_data=prompt_data,
        db_session=db_session
    )
    return DataResponse(
        data={
            'prompt': PromptOutScheme.model_validate(prompt)
        }
    )


@router.delete(
    '/{prompt_id}/',
    response_model=BaseResponse
)
async def delete_prompt(
        prompt: StylePrompt = Depends(get_prompt),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin]))
):
    await PromptRepo.delete(
        prompt=prompt,
        db_session=db_session
    )
    return BaseResponse(message    ='     ')
```

```
//src.routers.sessions.views.py
@router.get(
    '/',
    response_model=ListResponse[SessionOutScheme],
    responses=get_responses(400, 401)
)
async def get_sessions(
        user_info: UserInfo = Depends(JWTCookie()),
        db_session: AsyncSession = Depends(get_session),
        pagination: PaginationParams = Depends(get_pagination_params)
):
    sessions, count = await SessionRepo.get_list(
        user_id=user_info.id,
        pagination_params=pagination,
        db_session=db_session
    )
    return ListResponse[SessionOutScheme].from_list(
        items=sessions,
        total_count=count,
        params=pagination
    )


@router.post(
    '/close/',
    response_model=BaseResponse,
    responses=get_responses(400, 401)
)
async def close_sessions(
        user_info: UserInfo = Depends(JWTCookie()),
        db_session: AsyncSession = Depends(get_session),
):
    refresh_token_ids = await SessionRepo.get_refresh_token_ids(
        user_id=user_info.id,
        db_session=db_session
    )
    await put_tokens_in_black_list(refresh_token_ids)
    await SessionRepo.close_all(
        user_id=user_info.id,
        db_session=db_session
    )
    return BaseResponse(message ='                ')


//src.routers.reports.views.py
@router.post(
    '/articles/{article_id}/report/',
    response_model=DataResponse.single_by_key(
        'report',
        ReportOutScheme
    ),
```

```
    responses=get_responses(400, 401, 403, 409)
)
async def create_report(
        report_data: CreateReportScheme,
        report: Report | None = Depends(get_report(owner_only=True)),
        article_id: uuid.UUID = Path(),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user])),
):
    article = await ArticleRepo.get_by_id(
        article_id=article_id,
        db_session=db_session
    )
    if article.original_article_id is None:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail     ='                '
        )
    report = await ReportRepo.create(
        article_id=article_id,
        report_data=report_data,
        db_session=db_session
    )
    return DataResponse(
        data={
            'report': ReportOutScheme.create(report)
        }
    )



@router.put(
    '/articles/{article_id}/report/',
    response_model=DataResponse.single_by_key(
        'report',
        ReportOutScheme
    ),
    responses=get_responses(400, 401, 403, 404)
)
async def update_report(
        report_data: EditReportScheme,
        report: Report | None = Depends(get_report(owner_only=True)),
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user])),
):
    if not report:
        raise report_not_found_error
    report = await ReportRepo.update(
        report=report,
        report_data=report_data,
```

```
        db_session=db_session
    )
    return DataResponse(
        data={
            'report': ReportOutScheme.create(report)
        }
    )



@router.patch(
    '/articles/{article_id}/report/status/',
    response_model=DataResponse.single_by_key(
        'report',
        ReportOutScheme
    ),
    responses=get_responses(400, 401, 403, 404)
)
async def update_report_status(
        new_status: ReportStatus,
        article_id: uuid.UUID = Path(),
        report: Report | None = Depends(get_report(owner_only=False))
            ,
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[
            Role.user, Role.moderator
        ])),
):
    if not report:
        raise report_not_found_error
    if report.status != ReportStatus.open:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
            detail    ='       '
        )
    if (
        user_info.role == Role.user and new_status != ReportStatus.
           closed or
        user_info.role == Role.moderator and new_status not
        in [ReportStatus.rejected, ReportStatus.satisfied]
    ):
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN,
            detail    ='      '
        )
    return DataResponse(
        data={
            'report': ReportOutScheme.create(
                await ReportRepo.update_status(
                    report=report,
```

```
                    new_status=new_status,
                    user_id=user_info.id,
                    db_session=db_session
                )
            )
        }
    )


@router.get(
    '/articles/{article_id}/report/comments/',
    response_model=SimpleListResponse[CommentOutScheme],
    responses=get_responses(400, 401, 403, 409)
)
async def get_comments(
        report: Report | None = Depends(get_report(owner_only=False))
            ,
        user_info: UserInfo = Depends(JWTCookie(roles=[
            Role.user, Role.moderator
        ])),
        db_session: AsyncSession = Depends(get_session)
):
    if not report:
        raise report_not_found_error
    return SimpleListResponse[CommentOutScheme].from_list(
        await ReportRepo.get_comments(
            article_id=report.article_id,
            db_session=db_session
        )
    )


@router.post(
    '/articles/{article_id}/report/comments/',
    response_model=DataResponse.single_by_key(
        'comment',
        CommentOutScheme
    ),
    responses=get_responses(400, 401, 403, 404)
)
async def create_comment(
        comment_data: CreateCommentScheme,
        report: Report | None = Depends(get_report(owner_only=False))
            ,
        user_info: UserInfo = Depends(JWTCookie(roles=[
            Role.user, Role.moderator
        ])),
        db_session: AsyncSession = Depends(get_session)
):
    if not report or report.status != ReportStatus.open:
        raise report_not_found_error
```

```
    comment = await ReportRepo.create_comment(
        report_id=report.id,
        sender_id=user_info.id,
        text=comment_data.text,
        db_session=db_session
    )
    await db_session.refresh(report)
    redis_client = RedisHandler().client
    comment_scheme = CommentOutScheme(
            text=comment.text,
            sender_id=str(comment.sender_id),
            sender_name=(await UserRepo.get_by_id(
                user_id=user_info.id,
                db_session=db_session
            )).name,
            created_at=comment.created_at
        )
    await redis_client.publish(
        f'comments_{str(report.article_id)}',
        comment_scheme.model_dump_json()
    )
    return DataResponse(
        data={
            'comment': comment_scheme
        }
    )


//src.routers.translation.views.py
@router.post(
    '/',
    response_model=BaseResponse,
    responses=get_responses(400, 401, 403, 404)
)
async def create_translation(
        translation_data: CreateTranslationScheme,
        db_session: AsyncSession = Depends(get_session),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.user]))
):
    article = await ArticleRepo.get_by_id(
        article_id=translation_data.article_id,
        db_session=db_session
    )
    if not article or article.user_id != user_info.id:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail    ='       '
        )
    if article.original_article_id:
        raise HTTPException(
            status_code=status.HTTP_400_BAD_REQUEST,
```

```
            detail   ='           '
        )
    if not await ModelRepo.exists_by_id(
        model_id=translation_data.model_id,
        db_session=db_session
    ):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail   ='         '
        )
    if not await PromptRepo.exists_by_id(
        prompt_id=translation_data.prompt_id,
        db_session=db_session
    ):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail   ='         '
        )


    producer = KafkaProducer(
        bootstrap_servers=KafkaConfig.address,
        topic=KafkaConfig.translation_topic
    )


    for target_language_id in translation_data.target_language_ids:
        if not await LanguageRepo.exists(
                language_id=target_language_id,
                db_session=db_session
        ):
            raise HTTPException(
                status_code=status.HTTP_404_NOT_FOUND,
                detail    ='              '
            )
        task = await TaskRepo.create(
            task_data=CreateTaskScheme(
                article_id=translation_data.article_id,
                model_id=translation_data.model_id,
                prompt_id=translation_data.prompt_id,
                target_language_id=target_language_id
            ),
            db_session=db_session
        )
        message = TranslationMessage(task_id=task.id)
        await producer.send_message(
            message.model_dump(mode='json')
        )
        return BaseResponse(message    ='     .      ')
```

```
//src.routers.users.views.py
@router.patch(
    '/{user_id}/name/',
    response_model=BaseResponse,
    responses=get_responses(400, 401, 409)
)
async def change_name(
        request_data: UserUpdateNameScheme,
        user_id: uuid.UUID = Path(),
        user_info: UserInfo = Depends(JWTCookie()),
        db_session: AsyncSession = Depends(get_session)
):
    user = await UserRepo.get_by_id(
        user_id=user_info.id,
        db_session=db_session
    )
    if not user or user_id != user_info.id:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail      ='      '
        )
    if user.name == request_data.name:
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail  ='                    '
        )
    user.name = request_data.name
    db_session.add(user)
    await db_session.commit()
    return BaseResponse(message ='          ')


@router.post(
    '/',
    response_model=DataResponse.single_by_key(
        'user',
        UserOutScheme
    ),
    responses=get_responses(400, 401, 403, 409)
)
async def create_user(
        new_user_data: CreateUserScheme,
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin])),
        db_session: AsyncSession = Depends(get_session)
):
    user = await UserRepo.create(
        user_data=new_user_data,
        db_session=db_session
    )
```

```
    return DataResponse(
        data={
            'user': UserOutAdminScheme.model_validate(user)
        }
    )


@router.put(
    '/{user_id}/',
    response_model=DataResponse.single_by_key(
        'user',
        UserOutScheme
    ),
    responses=get_responses(400, 401, 403, 409)
)
async def update_user(
        new_user_info: EditUserScheme,
        user: User = Depends(get_user),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin])),
        db_session: AsyncSession = Depends(get_session),
):
    user = await UserRepo.update(
        user=user,
        new_data=new_user_info,
        db_session=db_session
    )
    return DataResponse(
        data={
            'user': UserOutAdminScheme.model_validate(user)
        }
    )


@router.delete(
    '/{user_id}/',
    responses=get_responses(400, 401, 403, 409)
)
async def delete_user(
        user: User = Depends(get_user),
        user_info: UserInfo = Depends(JWTCookie(roles=[Role.admin])),
        db_session: AsyncSession = Depends(get_session),
):
    await UserRepo.soft_delete(
        user=user,
        db_session=db_session
    )
    return BaseResponse(message      ='     ')
```

```
//src.database.models.py
class User(Base):
    __tablename__ = f'{Database.prefix}users'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    name: Mapped[str] = mapped_column(
        String(20)
    )
    email: Mapped[str] = mapped_column(
        String,
        unique=True
    )
    email_verified: Mapped[bool] = mapped_column(
        Boolean,
        default=False
    )
    password_hash: Mapped[str] = mapped_column(
        String(60)
    )
    role: Mapped[Role] = mapped_column(
        Enum(Role, name='user_role'),
        default=Role.user
    )
    logged_with_provider: Mapped[str | None] = mapped_column(
        String,
        nullable=True,
        comment='External OAuth provider name user has registered
            with'
    )
    provider_id: Mapped[str | None] = mapped_column(
        String,
        nullable=True,
        comment='User\'s ID from OAuth provider user has registered
            with'
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )
    deleted_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )



class Session(Base):
```

```
    __tablename__ = f'{Database.prefix}sessions'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    user_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{Database.prefix}users.id', ondelete='CASCADE')
    )
    ip: Mapped[str] = mapped_column(
        String(15)
    )
    user_agent: Mapped[str] = mapped_column(
        String(100)
    )
    is_closed: Mapped[bool] = mapped_column(
        Boolean,
        default=False
    )
    refresh_token_id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )
    closed_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )



class ConfirmationCode(Base):
    __tablename__ = f'{Database.prefix}confirmation_codes'
    id: Mapped[int] = mapped_column(
        Integer,
        primary_key=True
    )
    code: Mapped[str] = mapped_column(
        String,
        unique=True,
        comment='The value of the code'
    )
    reason: Mapped[ConfirmationType] = mapped_column(
        Enum(ConfirmationType),
        default=ConfirmationType.registration
    )
    user_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{User.__tablename__}.id', ondelete='CASCADE')
```

```
    )
    expired_at: Mapped[datetime.datetime] = mapped_column(
        DateTime
    )
    is_used: Mapped[bool] = mapped_column(
        Boolean,
        default=False
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )




class Language(Base):
    __tablename__ = f'{Database.prefix}languages'
    id: Mapped[int] = mapped_column(
        Integer,
        primary_key=True
    )
    name: Mapped[str] = mapped_column(
        String,
        unique=True
    )
    iso_code: Mapped[str] = mapped_column(
        String,
        unique=True
    )




class Article(Base):
    __tablename__ = f'{Database.prefix}articles'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    title: Mapped[str] = mapped_column(String(50))
    text: Mapped[str] = mapped_column(Text)
    user_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{User.__tablename__}.id', ondelete='CASCADE')
    )
    language_id: Mapped[int | None] = mapped_column(
        ForeignKey(f'{Language.__tablename__}.id', ondelete='CASCADE
            '),
        nullable=True
    )
```

```
    original_article_id: Mapped[uuid.UUID | None] = mapped_column(
        ForeignKey(f'{Database.prefix}articles.id', ondelete='CASCADE
            '),
        nullable=True
    )
    like: Mapped[bool | None] = mapped_column(
        Boolean,
        nullable=True
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )
    deleted_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )


    report: Mapped['Report'] = relationship(
        'Report',
        back_populates='article',
        cascade='all, delete-orphan',
        uselist=False,
        lazy='joined'
    )
    language: Mapped[Language] = relationship(
        'Language',
        uselist=False,
        lazy='joined'
    )
    original_article: Mapped['Article'] = relationship(
        'Article',
        uselist=False,
        lazy='joined'
    )




class ReportReason(Base):
    __tablename__ = f'{Database.prefix}report_reasons'
    id: Mapped[int] = mapped_column(
        Integer,
        primary_key=True
    )
    text: Mapped[str] = mapped_column(
        String,
        unique=True
    )
    order_position: Mapped[int] = mapped_column(
```

```
        Integer,
        unique=True
    )



class Report(Base):
    __tablename__ = f'{Database.prefix}reports'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    text: Mapped[str] = mapped_column(
        String(1024)
    )
    article_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{Article.__tablename__}.id', ondelete='CASCADE')
    )
    status: Mapped[ReportStatus] = mapped_column(
        Enum(ReportStatus),
        default=ReportStatus.open
    )
    closed_by_user_id: Mapped[uuid.UUID | None] = mapped_column(
        ForeignKey(f'{User.__tablename__}.id', ondelete='CASCADE'),
        nullable=True
    )
    reason_id: Mapped[int] = mapped_column(
        ForeignKey(f'{ReportReason.__tablename__}.id', ondelete='
            CASCADE')
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )
    closed_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )


    article: Mapped[Article] = relationship(
        'Article',
        back_populates='report',
        uselist=False,
        lazy='joined',
    )
    closed_by_user: Mapped[User] = relationship(
        'User',
        uselist=False,
```

```
        lazy='joined',
    )
    reason: Mapped[ReportReason] = relationship(
        'ReportReason',
        uselist=False,
        lazy='joined',
    )



class Comment(Base):
    __tablename__ = f'{Database.prefix}report_comments'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    text: Mapped[str] = mapped_column(
        String(100)
    )
    sender_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{User.__tablename__}.id', ondelete='CASCADE')
    )
    report_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{Report.__tablename__}.id', ondelete='CASCADE')
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )



class StylePrompt(Base):
    __tablename__ = f'{Database.prefix}style_prompts'
    id: Mapped[int] = mapped_column(
        Integer,
        primary_key=True
    )
    title: Mapped[str] = mapped_column(
        String(20),
        unique=True
    )
    text: Mapped[str] = mapped_column(
        String,
        unique=True
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
```

```
        default=get_utc_now
    )
    deleted_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )



class AIModel(Base):
    __tablename__ = f'{Database.prefix}ai_models'
    id: Mapped[int] = mapped_column(
        Integer,
        primary_key=True
    )
    show_name: Mapped[str] = mapped_column(String(50), nullable=False
        )
    name: Mapped[str] = mapped_column(String, nullable=False)
    provider: Mapped[str] = mapped_column(String, nullable=False)
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now,
        nullable=False
    )
    deleted_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )



class TranslationConfig(Base):
    __tablename__ = f'{Database.prefix}configs'
    id: Mapped[int] = mapped_column(
        Integer,
        primary_key=True
    )
    user_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{User.__tablename__}.id', ondelete='CASCADE')
    )
    prompt_id: Mapped[int | None] = mapped_column(
        ForeignKey(
            f'{StylePrompt.__tablename__}.id',
            ondelete='CASCADE'
        ),
        nullable=True
    )
    name: Mapped[str] = mapped_column(
        String(20),
```

```
    )
    language_ids: Mapped[list[int]] = mapped_column(ARRAY(Integer))
    model_id: Mapped[int | None] = mapped_column(
        ForeignKey(f'{AIModel.__tablename__}.id', ondelete='CASCADE')
            ,
        nullable=True
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )
    deleted_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )



class TranslationTask(Base):
    __tablename__ = f'{Database.prefix}translation_tasks'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    article_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{Article.__tablename__}.id', ondelete='CASCADE')
    )
    target_language_id: Mapped[int] = mapped_column(
        ForeignKey(f'{Language.__tablename__}.id', ondelete='CASCADE
            ')
    )
    prompt_id: Mapped[int] = mapped_column(
        ForeignKey(
            f'{StylePrompt.__tablename__}.id', ondelete='CASCADE'
        )
    )
    model_id: Mapped[int] = mapped_column(
        ForeignKey(
            f'{AIModel.__tablename__}.id', ondelete='CASCADE'
        )
    )
    status: Mapped[TranslationTaskStatus] = mapped_column(
        Enum(TranslationTaskStatus),
        default=TranslationTaskStatus.created
    )
    data: Mapped[dict] = mapped_column(
        JSONB,
        nullable=True,
        comment='Additional data related to the translation task '
```

```
                '(e.g., errors or metadata)'
    )
    translated_article_id: Mapped[uuid.UUID | None] = mapped_column(
        ForeignKey(f'{Article.__tablename__}.id', ondelete='CASCADE')
            ,
        nullable=True
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )
    deleted_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )




class Notification(Base):
    __tablename__ = f'{Database.prefix}notifications'
    id: Mapped[uuid.UUID] = mapped_column(
        UUID(as_uuid=True),
        primary_key=True,
        default=uuid.uuid4
    )
    title: Mapped[str] = mapped_column(
        String
    )
    text: Mapped[str] = mapped_column(
        String
    )
    user_id: Mapped[uuid.UUID] = mapped_column(
        ForeignKey(f'{User.__tablename__}.id', ondelete='CASCADE')
    )
    type: Mapped[NotificationType] = mapped_column(
        Enum(NotificationType)
    )
    read_at: Mapped[datetime.datetime | None] = mapped_column(
        DateTime,
        nullable=True
    )
    created_at: Mapped[datetime.datetime] = mapped_column(
        DateTime,
        default=get_utc_now
    )




//tests.hot_load.py
```

```
import asyncio
import datetime
import logging
import multiprocessing
import random
import statistics
import time
from collections import defaultdict
from functools import wraps


import httpx
logger = logging.getLogger(__name__)


class HotLoad:
    def __init__(
            self,
            duration: datetime.timedelta,
            processes_number: int = 1,
            workers_number: int = 1,
    ):
        self.duration = duration
        self.deadline = datetime.datetime.now() + duration
        self.processes_number = processes_number
        self.workers_number = workers_number
        self.headers = {}
        self.tasks = []
        self.errors = 0  # find usage
        self.on_startup_callable = None
        self.on_teardown_callable = None
    def task(self, func):
        @wraps(func)
        async def wrapper(*args, **kwargs):
            return await func(*args, **kwargs)
        self.tasks.append(wrapper)
        return wrapper
    def on_startup(self, func):
        @wraps(func)
        async def wrapper(*args, **kwargs):
            return await func(*args, **kwargs)
        self.on_startup_callable = wrapper
        return wrapper
    def on_teardown(self, func):
        @wraps(func)
        async def wrapper(*args, **kwargs):
            return await func(*args, **kwargs)
        self.on_teardown_callable = wrapper
        return wrapper
    @staticmethod
    def get_median(results) -> float:
```

```
    return statistics.median(results) if results else 0
@staticmethod
def get_timestamp_now():
    return datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
async def run_worker(
        self,
        worker_id: int
) -> int:
    logger.info('Running worker %s', worker_id)
    stats = defaultdict(list)
    total_requests = 0
    async with httpx.AsyncClient(headers=self.headers) as client:
        while self.deadline > datetime.datetime.now():
            try:
                start = time.time()
                await self.tasks[
                    random.randint(0, len(self.tasks) - 1)
                ](
                    client,
                    worker_id
                )
                end = time.time()
                delta = end - start
                current_timestamp = self.get_timestamp_now()
                stats[current_timestamp].append(delta)
            except Exception as e:
                logger.exception(e)
                self.errors += 1
            total_requests += 1
    return total_requests
async def run_workers(self, worker_start_id: int) -> int:
    results = await asyncio.gather(*[
        self.run_worker(worker_id=i + worker_start_id)
        for i in range(self.workers_number)
    ])
    return sum(results)
def run_process(self, process_number: int, *args) -> int:
    worker_start_id = process_number * self.workers_number
    loop = asyncio.get_event_loop()
    result = loop.run_until_complete(self.run_workers(
       worker_start_id))
    return result


async def run(self) -> float:
    if self.on_startup_callable:
        self.headers = await self.on_startup_callable()


    with multiprocessing.Pool(processes=self.processes_number) as
        pool:
```

```
        results = pool.map(self.run_process, range(self.
            processes_number))
    mean_rps = sum(results) / self.duration.total_seconds()
    if self.on_teardown_callable:
        await self.on_teardown_callable()
    return mean_rps
```

Приложение Б

Скрипт создания объектов базы данных

```
    create database diploma    with owner admin;
create type public.user_role as enum ('user', 'moderator', 'admin');
alter type public.user_role owner to admin;
create type public.confirmationtype as enum ('registration', '
   password_reset');
alter type public.confirmationtype owner to admin;
create type public.notificationtype as enum ('info', 'success', '
   warning', 'error');
alter type public.notificationtype owner to admin;
create type public.reportstatus as enum ('open', 'closed', 'rejected
   ', 'satisfied');
alter type public.reportstatus owner to admin;
create type public.translationtaskstatus as enum ('created', 'started
   ', 'failed', 'completed');
alter type public.translationtaskstatus owner to admin;
create table public.alembic_version
(    version_num varchar(32) not null
        constraint alembic_version_pkc
            primary key
);
alter table public.alembic_version    owner to admin;
create table public.gptranslate_ai_models
(    id          serial
        primary key,
    show_name  varchar(50) not null,
    name       varchar     not null,
    provider   varchar     not null,
    created_at timestamp   not null,
    deleted_at timestamp
);


alter table public.gptranslate_ai_models
    owner to admin;


create table public.gptranslate_languages
(
    id       serial
        primary key,
    name     varchar not null
        unique,
    iso_code varchar not null
        unique
);


alter table public.gptranslate_languages
```

```
    owner to admin;


create table public.gptranslate_report_reasons
(
    id              serial
        primary key,
    text            varchar not null
        unique,
    order_position integer not null
        unique
);


alter table public.gptranslate_report_reasons
    owner to admin;


create table public.gptranslate_style_prompts
(
    id          serial
        primary key,
    title       varchar(20) not null
        unique,
    text        varchar     not null
        unique,
    created_at timestamp   not null,
    deleted_at timestamp
);


alter table public.gptranslate_style_prompts
    owner to admin;


create table public.gptranslate_users
(
    id                    uuid        not null
        primary key,
    name                  varchar(20) not null,
    email                 varchar     not null
        unique,
    email_verified        boolean     not null,
    password_hash         varchar(60) not null,
    role                  user_role   not null,
    logged_with_provider varchar,
    provider_id           varchar,
    created_at            timestamp   not null,
    deleted_at            timestamp
);
```

```
comment on column public.gptranslate_users.logged_with_provider is '
   External OAuth provider name user has registered with';


comment on column public.gptranslate_users.provider_id is 'User''s ID
    from OAuth provider user has registered with';

alter table public.gptranslate_users
    owner to admin;


create table public.gptranslate_articles
(
    id                  uuid        not null
        primary key,
    title               varchar(50) not null,
    text                text        not null,
    user_id             uuid        not null
        references public.gptranslate_users
            on delete cascade,
    language_id         integer
        references public.gptranslate_languages
            on delete cascade,
    original_article_id uuid
        references public.gptranslate_articles
            on delete cascade,
    "like"              boolean,
    created_at          timestamp   not null,
    deleted_at          timestamp
);


alter table public.gptranslate_articles
    owner to admin;


create table public.gptranslate_configs
(
    id           serial
        primary key,
    user_id      uuid        not null
        references public.gptranslate_users
            on delete cascade,
    prompt_id    integer
        references public.gptranslate_style_prompts
            on delete cascade,
    name         varchar(20) not null,
    language_ids integer[]   not null,
    model_id     integer
        references public.gptranslate_ai_models
```

```
            on delete cascade,
    created_at   timestamp   not null,
    deleted_at   timestamp
);


alter table public.gptranslate_configs
    owner to admin;


create table public.gptranslate_confirmation_codes
(
    id          serial
        primary key,
    code        varchar          not null
        unique,
    reason      confirmationtype not null,
    user_id     uuid             not null
        references public.gptranslate_users
            on delete cascade,
    expired_at timestamp         not null,
    is_used     boolean          not null,
    created_at timestamp         not null
);


comment on column public.gptranslate_confirmation_codes.code is 'The
   value of the code';


alter table public.gptranslate_confirmation_codes
    owner to admin;


create table public.gptranslate_notifications
(
    id          uuid             not null
        primary key,
    title       varchar          not null,
    text        varchar          not null,
    user_id     uuid             not null
        references public.gptranslate_users
            on delete cascade,
    type        notificationtype not null,
    read_at     timestamp,
    created_at timestamp         not null
);


alter table public.gptranslate_notifications
    owner to admin;
```

```
create table public.gptranslate_sessions
(
    id               uuid         not null
        primary key,
    user_id          uuid         not null
        references public.gptranslate_users
            on delete cascade,
    ip               varchar(15)  not null,
    user_agent       varchar(100) not null,
    is_closed        boolean      not null,
    refresh_token_id uuid         not null,
    created_at       timestamp    not null,
    closed_at        timestamp
);


alter table public.gptranslate_sessions
    owner to admin;


create table public.gptranslate_reports
(
    id                uuid          not null
        primary key,
    text              varchar(1024) not null,
    article_id        uuid          not null
        references public.gptranslate_articles
            on delete cascade,
    status            reportstatus  not null,
    closed_by_user_id uuid
        references public.gptranslate_users
            on delete cascade,
    reason_id         integer       not null
        references public.gptranslate_report_reasons
            on delete cascade,
    created_at        timestamp     not null,
    closed_at         timestamp
);


alter table public.gptranslate_reports
    owner to admin;
create table public.gptranslate_translation_tasks
(
    id                    uuid                    not null
        primary key,
    article_id            uuid                    not null
        references public.gptranslate_articles
            on delete cascade,
```

```
    target_language_id    integer                not null
        references public.gptranslate_languages
            on delete cascade,
    prompt_id             integer                not null
        references public.gptranslate_style_prompts
            on delete cascade,
    model_id              integer                not null
        references public.gptranslate_ai_models
            on delete cascade,
    status                translationtaskstatus not null,
    data                  jsonb,
    translated_article_id uuid
        references public.gptranslate_articles
            on delete cascade,
    created_at            timestamp              not null,
    deleted_at            timestamp
);
comment on column public.gptranslate_translation_tasks.data is '
   Additional data related to the translation task (e.g., errors or
   metadata)';
alter table public.gptranslate_translation_tasks
    owner to admin;
create table public.gptranslate_report_comments
(
    id         uuid         not null
        primary key,
    text       varchar(100) not null,
    sender_id  uuid         not null
        references public.gptranslate_users
            on delete cascade,
    report_id  uuid         not null
        references public.gptranslate_reports
            on delete cascade,
    created_at timestamp    not null
);
alter table public.gptranslate_report_comments
    owner to admin;
```