МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

ИНТЕЛЛЕКТУАЛЬНАЯ ДИАЛОГОВАЯ СИСТЕМА ВУЗА

Студент _____ 27.05.2024 Митрофанов Максим Павлович
(подпись, дата) (ФИО полностью)

Руководитель _____ 27.05.2024 Карамышева Н. С.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ 27.05.2024 Бычков А. С.
(подпись, дата) (фамилия, инициалы)

Рецензент _____ Мартышкин А.М., к.т.н., зав. каф. "Программирование" ПензГТУ
(подпись, дата) (фамилия, инициалы) (должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой _____ Митрохин М. А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 06.06.24 № 8)

Секретарь ГЭК _____ Гурин Е. И.
(подпись) (фамилия, инициалы)

Пенза 2024

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М. А. Митрохин
«13» сентября 2022г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

1.Студент Митрофанов Максим Павлович гр. 22ВВИм1 факультета вычислительной техники, направления 09.04.01.

2. Тема работы «Интеллектуальная диалоговая система вуза».

Тема утверждена приказом ПГУ № 765/о от "13"сентября 2022 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры ВТ ПГУ Карамышева Н. С.

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

Назначение разработки: создание интеллектуальной диалоговой системы вуза для помощи студентам и абитуриентам.

Разработать автоматизированное клиент-серверное программное обеспечение с использованием искусственного интеллекта для предоставления информации по заданным датасетам.

5. Перечень подлежащих разработке вопросов:

1) Анализ уже действующих решений из сферы нейронных сетей.
2) Создание базы данных.
3) Разработка интеллектуальной диалоговой системы вуза на основе клиент-серверной архитектуры.

4) Тестирование системы согласно ранее описанному техническому заданию на разработку.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучить теоретические основы	10%	27.09.2022-15.11.2023	
2.	Анализ технических заданий, анализ существующих ИИ-ассистентов	10%	16.11.2023-22.03.2023	
3.	Выбор инструментов для разработки и утверждение функционала системы	5%	23.03.2023-14.06.2023	
4.	Спроектировать базу данных	10%	15.06.2023-10.09.2023	
5.	Написать логику работы приложения	40%	11.09.2023-05.01.2024	
6.	Интегрировать ИИ-ассистента и реализовать подключение собственного датасета	15%	06.01.2024-17.04.2024	
7.	Апробация разработанного клиент-серверного приложения в форме тестирования	5%	18.04.2024-9.05.2024	
8.	Оформление выпускной работы	5%	10.05.2024-28.05.2024	

Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации _____ 13.09.2022 Карамышева Н.С.
(подпись, дата)

Задание к исполнению принял студент _____ 13.09.2022 Митрофанов М. П.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 03.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 159 листа, 27 рисунков, 8 источников, 2 приложения

AIOGRAM, SQLALCHEMY, PYTHON, PYCHARM IDE, ALEMBIC, CHATGPT, OPENAI, CHATSONIC, ЧАТ-БОТЫ, ОБРАБОТКА СОСТОЯНИЙ, ПОДАЧА ДОКУМЕНТОВ, ПОДАЧА ЗАЯВКИ, БАЗА ДАННЫХ, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ, НЕЙРОННЫЕ СЕТИ

Цель работы – необходимо разработать интеллектуальную диалоговую систему вуза для оперативной круглосуточной помощи студентам и абитуриентам как с организационной деятельностью вуза, так и с выбором вуза.

В результате проведенной работы было разработано программное обеспечение, позволяющее автоматизировать ответы различных преподавателей и сделать их круглосуточными за счет интеграции с искусственным интеллектом, позволяющие использовать конкретный датасет для использования искусственным интеллектом.

В ходе разработки интеллектуальной диалоговой системы вуза, использовались следующие технологии: python, aiogram, sqlalchemy, alembic, sqlite, telegram API, ChatGPT.

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 5

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 6

1 Анализ технического задания 10

1.1 Анализ функциональных требований 10

1.2 Анализ нефункциональных требований 11

2 Анализ предметной области 13

2.1 Решение поставленной задачи 14

2.2 Анализ классификаций чат-ботов 16

2.3 Анализ и структура бота 18

2.4 Анализ видов нейронных сетей 21

2.5 Постановка задачи 27

2.6 Описание архитектуры проекта 28

2.7 Выбор инструментов разработки 29

2.8 Выбранные технологии 35

3 Реализация 36

3.1 Регистрация чат-бота 36

3.2 Работа с БД 37

3.3 Описание доступных функций 42

3.4 Описание программы 48

ЗАКЛЮЧЕНИЕ 65

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 67

ПРИЛОЖЕНИЕ А Листинг программы 68

ПРИЛОЖЕНИЕ Б Презентация 150

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

БД – база данных

ИИ – искусственный интеллект

ПК – персональный компьютер

СНГ – содружество независимых государств

ССС – совет студенческого самоуправления

СУБД – система управления базами данных

ЯП – язык программирования

ACID (Atomicity, Consistency, Isolation, Durability) – требования к транзакционной системе, обеспечивающие наиболее надёжную и предсказуемую её работу

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

GPT (Generative Pre-trained Transformer) – генеративный, предобученный трансформер

IDE (Integrated development environment) – интегрированная среда разработки

ID (Identifier) – идентификатор объекта

JSON (JavaScript Object Notation) – текстовый формат обмена данными

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

ВВЕДЕНИЕ

Один из самых популярных способов коммуникаций с пользователями в настоящее время – это чат-боты. Их популярность заключается в простоте разработки, интеграции в действующее системы крупных учебных учреждений и организаций, а также легкостью в использовании и обучении конечного потребителя.

Диалоговые системы продолжают развиваться и по сей день. Большинство современных чат-ботов реализованы на базе искусственного интеллекта, который заменяет устаревшие методы поиска информации, “живых” операторов ПК, а также отмечается молниеносной скоростью ответа.

Интеллектуальная диалоговая система вуза обуславливает свою актуальность современными технологиями реализации, которые сильно удешевляют последующее использование чат-бота, а также лишают необходимости работы “живых” операторов на протяжении полного дня для молниеносных ответов пользователю согласно многочисленным исследованиям.

В сфере образования, как и в прочих сферах жизнедеятельности использование диалоговых интеллектуальных систем является актуальным решением. Мгновенная обратная связь, получаемая абитуриентом от ИИ-ассистента по рейтингу вузов, своевременная помощь студентам с учебной программой, получение и отправка документов в деканат– минимально-жизнеспособный проект интеллектуальной диалоговой системы вуза, который подлежит дальнейшему развитию в рамках прикладного искусственного интеллекта.

Одним из инновационных решений является использование искусственного интеллекта. С его помощью диалоговая система сможет быстро ориентироваться как в сети интернет, так и на основе выбранных наборов данных.

Целями магистерской диссертации являются:

– интеграция ИИ в диалоговую систему,

– обучение ИИ на основе подготовленных наборов данных,

– изучение актуальных подходов к реализации чат-диалоговых систем на базе мессенджеров,

– реализация чат-бота с использованием ИИ-ассистента для автоматизации процесса получения информации о месте вуза в мировом рейтинге, а также для его последующего применения в деятельности совета студенческого самоуправления.

Для достижения поставленных целей, необходимо обозначить следующие задачи, решаемые в ходе выполнения магистерской диссертации:

– провести анализ действующих процессов деятельности совета студенческого самоуправления,

– поставить задачи,

– провести анализ существующих чат-ботов,

– провести анализ существующих ИИ,

– выбрать наиболее современный способ обучения ИИ работе с наборами данных.

– провести анализ технического задания на создание интеллектуальной диалоговой системы,

– провести анализ архитектуры бота,

– спроектировать карту действий бота,

– реализовать интеллектуальную диалоговую систему,

– протестировать интеллектуальную диалоговую систему.

В соответствии с поставленными задачами можно выделить четыре основных раздела данной магистерской работы.

В первом разделе происходит анализ технического задания, описывается выбор решений для разработки интеллектуальной диалоговой системы с учетом заявленных в магистерской работе функциональных и нефункциональных требований.

Во втором разделе предоставлен подробный анализ существующих решений в сфере диалоговых систем, а также анализ существующих готовых решений в сфере искусственного интеллекта, в частности – использование искусственного интеллекта с целью получения текстовых ответов, составленных на основе наборов данных. В конце раздела будет проведена сравнительная характеристика существующих решений.

В третьем разделе будет описана реализация интеллектуальной диалоговой системы вуза с учётом проведенного исследования и анализа во втором разделе.

В четвёртом разделе будет находиться описание методов и подходов тестирования объекта магистерской работы.

Итогом выполнения поставленных задач данной магистерской диссертации будет реализована собственная интеллектуальная диалоговая система вуза, реализованная на основе выбранных решений, найденных путем проведения исследований и анализа предметной области.

1 Анализ технического задания

В рамках данного раздела необходимо проанализировать и выбрать те функциональные и нефункциональные требования, необходимые к реализации для достижения целей проекта.

1.1 Анализ функциональных требований

Интеллектуальная диалоговая система вуза – программное обеспечение, реализуемое с целью помочь абитуриентам при поступлении в наиболее подходящие для них вузы с помощью ИИ-ассистента, а также помочь действующим студентам университетов и колледжей проводить определенный список операций, который оптимизирует их деятельность в рамках совета студенческого самоуправления.

Кроме того, используемый ИИ-ассистент сможет отвечать на поставленные вопросы по учебной программе действующим студентам.

Основными функциональными требованиями данного ПО являются:

– возможность регистрации в чат-боте посредством заполнения анкеты,

– просмотр новостной ленты факультетов, студенческих организацией,

– возможность вручную создавать посты с текстовой информацией и изображениями для новостной ленты,

– возможность загрузить шаблоны документов в формате PDF от имени администратора,

– возможность скачивать ранее загруженные шаблоны документов в формате PDF от имени пользователя,

– возможность отображения базы данных телефонных номеров активистов прямо в диалоговой системе,

– использование ИИ-ассистента для оперативного реагирования на запросы от пользователей по части учебной программы,

– возможность выдачи ответов на вопросы ИИ-ассистентом и его обучение на основе набора данных в формате CSV.

1.2 Анализ нефункциональных требований

Нефункциональные требования главным образом предъявляются к аппаратному обеспечению компьютера и производительности. Таким образом, для функционирования данной системы требуется любое устройство, поддерживающее мессенджер “Telegram”, а также имеет доступ к сети интернет.

1.2.1 Требования к организации входных данных

Чат-бот получает данные для обработки от пользователей двух ролей – активист и администратор. Вводные данные должны быть на русском языке, номер телефона вводиться с “8”. Файлы, загружаемые в качестве шаблонов – не более 10мб, в формате PDF. ИИ-ассистент должен реагировать на текстовые сообщения пользователя. Набор данных для реализации базы знаний для ИИ-ассистента должен представлять собой CSV-файл.

1.2.2 Требования к организации выходных данных

Выходным данными для роли “администратор” являются данные из базы данных (номера телефонов и ФИО активистов), которые выводятся текстовым сообщением. Для роли “Активист” выходными данными также являются данные из БД, ранее загруженные “администратором” – новостные посты, шаблоны документов в формате PDF. Кроме того, ИИ-ассистент генерирует ответы на вопросы пользователя как на основе собственной базы знаний, так и на основе выделенного набора данных в формате CSV.

Требования к организации выходных данных:

– данные, получаемые пользователем должны быть достоверны и быть на русском языке,

– интеллектуальная диалоговая система должна корректно и полно отвечать на запросы пользователя,

– информация, получаемая пользователем, должна поступать последовательно и логично,

– система должна быть отзывчивой на всех возможных носителях, таких как: ПК, Ноутбуки, Планшеты, смартфоны и тд.,

– выходные данные, получаемые пользователем, должны поступать в режиме реального времени за максимально быстрый срок.

1.2.3 Требования к организации тестирования

Тестирование - один из важнейших шагов в цикле разработки ПО, который отвечает за соответствие готового ПО функциональным, нефункциональным требованиям, которые утверждаются на этапе инициализации того или иного продукта.

Для того чтобы протестировать интеллектуальную диалоговую систему вуза, нам необходимо составить чек-лист тестирования, который содержит в себе максимально возможное количество проверок.

Был разработан следующий чек-лист тестирования:

– проверить что чат-бот принимаем на вход текстовые значения,

– проверить, что чат-бот корректно обрабатывает нажатия на клавиши

– проверить, бот корректно выводит базу данных номеров пользователей,

– проверить, что на электронную почту приходят заявки на вступление в совет студенческого самоуправления,

– проверить, что ИИ-ассистент корректно отвечает на вопросы, задаваемые пользователем,

– проверить, что ИИ-ассистент корректно отвечает на вопросы, задаваемые пользователем,

– проверить, ИИ-ассистент корректно обрабатывает вопросы, ответ на который содержится в подготовленном наборе данных,

– проверить, что администратор может создавать новостные посты,

– проверить, что пользователь может просматривать новостные посты.

2 Анализ предметной области

Перед началом разработки ПО необходимо провести ряд исследований и анализом, которые будут являться первым шагом инициации создания нового программного обеспечения, которое поможет выполнять пользователю те или иные поставленные функции. Необходимо понимать, по каким методикам и способам разрабатывать ПО, как будет выглядеть архитектура и т.д. В рамках данного раздела необходимо рассмотреть ряд теоретических моментов, на основе которых будет понятно, как максимально корректно реализовать интеллектуальную диалоговую систему вуза.

Первым этапом на пути к разработке чат-бота со встроенным ИИ-ассистентом будет анализ тех методов поиска информации и вариантов выполнения функций, выполняемыми нашей системой, которыми пользователи пользуются прямо сейчас.

Главным информационным полем, которое отвечает пользователям на большинство вопросов, является официальный сайт вуза. К сожалению, на сайте не всегда можно своевременно получить ту или иную информацию, не каждый раздел доработан, кроме того - пользователю, открывшему сайт в первый раз - будет трудно ориентироваться. Плюсами сайта является совокупность всей важной информации, которая может потребоваться и абитуриенту, и студенту, и преподавателям.

Метод, который наиболее популярен у целевой аудитории интеллектуальной диалоговой системы в возрасте от 14 до 30 - получение информации в различном формате с помощью социальных сетей. Уверенному пользователю ПК или смартфона достаточно легко с помощью поисковых алгоритмов соцсетей обнаружить группы, посвящённые как главным структурным подразделениям университета, так и каналы, посвященные внеурочной студенческой жизни, каналы факультетов, кружков и прочих ответвлений совета студенческого самоуправления. Однако, методом данного метода является сугубо информационный характер пабликов. Вы можете

знакомиться только с той информацией, что есть в группах, ждать ответа от администраторов групп чаще всего достаточно долго, невозможно произвести операции по подаче документов и тд.

Для родителей абитуриентов и студентов, как правило, одним из лучших решений для получения информации - живое общение с представителями факультетов, кафедр различных направлений, чтобы вживую задать все интересующие вопросы. В Пензенском Государственном университете реализована целая серия мероприятий “День открытых дверей”, на которых можно поинтересоваться у сотрудников университета и действующих студентов.

С недавних пор появился еще один канал информации, отличающийся от других скоростью ответа и полнотой информации - нейронные сети. Их использование повсеместно, они используются во всех сферах жизнедеятельности людей, в том числе и в сфере образования. Вы можете спросить у ИИ-ассистентов различного уровня те или иные вопросы, касающиеся университетов и жизни в них.

В настоящий момент на рынке нет инструмента с понятным для пользователя интерфейсом, позволяющего абитуриентам и студентам получать информацию со стороны вуза, пользоваться круглосуточной поддержкой по учебной программе. Кроме того, нет инструмента, автоматизирующего процессы совета студенческого самоуправления.

2.1 Решение поставленной задачи

Интеллектуальная диалоговая система - современное программное обеспечение, отвечающие на вопросы по студенческой тематике в режиме реального времени с использованием нейронных сетей.

Похожие чат-боты используются крупными коммерческими организациями, государственными учреждениями, в образовательных школах и прочих сферах жизнедеятельности. Наиболее близкими продуктами к

интеллектуальной диалоговой системе вуза разрабатываемой в рамках данной работы являются:

– бот-ассистент для абитуриента. Цель данного ПО - вовлечение абитуриентов на поступление в университет и продажа курсов от факультетов среди пользователей системы,

– чат-бот ИАТЭ. Целью данного бота является привлечение в ВУЗ новых студентов,

– бот ТГУ. Бот, которые помогает провести оптимизацию поиска данных с сайта вуза,

Плюсами чат-ботов выше являются:

– мгновенная скорость ответа. Бот работает в режиме реального времени, засчет чего отвечает сразу же,

– возможность использовать интеллектуальную диалоговую систему анонимно,

– нет необходимости записывать, сохранять или иным способам фиксировать необходимую информацию - диалог с ботом не пропадет, если его не удалить,

– пользователю не нужно рассматривать и изучать большой блок информации перед тем, как понять, какие действия необходимо выполнить для поиска информации. Весь интерфейс - это несколько клавиш и диалоговое окно,

– на основе исследований, люди лучше воспринимают данные если общаются с себе подобным ассистентом,

– возможность анализа аудитории для построения гипотез, которые в дальнейшем могут помочь с развитием и масштабированием проекта.

С использованием ИИ-ассистента в интеллектуальной диалоговой системе вуза - не требуются операторы, которые могли бы отвечать на вопросы пользователей. Кроме того пользователи могут использовать чат-бота круглосуточно и это не скажется на времени ответа. Кроме того, бот минимально зависит от человеческого фактора во время работы. У бота не

бывает плохого настроения, он не может сам изменить скрипт сообщения, что минимизирует грамматические ошибки либо неточные ответы пользователям.

2.2 Анализ классификаций чат-ботов

Прежде чем приступить к этапу разработки любого программного обеспечения, необходимо ответственно подойти к более ранним этапам, в нашем случае - к анализу алгоритмов чат-ботов, а также их классификации и виды. Необходим анализ для выбора наиболее подходящей архитектуры под цели проекта.

Боты, главным ядром которых, являются нейронные сети, называются “саморазвивающимися”. Преимущество такого бота в том, что с каждым произведенным диалогом с пользователем за счет своих алгоритмов обучения диалоговая система выдает ответы все более и более точно. Чат-бот учится поддерживать разговор чтобы максимально удовлетворить запросы клиента.

Боты, в основе которых лежат заранее прописанные скрипты аналитиками, заказчиками, или прочими людьми, влияющими на проект, встречаются чаще обычного за счет простоты разработки. Название такого вида диалоговых систем - ограниченные, в связи с тем, что пользователь может получить ответ лишь по тем вопросам, которые заранее определил владелец бота. В случае появления какого-либо вопроса, не предусмотренного скриптом - необходимо будет изменить метод коммуникации с организацией.

Другая классификация, по формату взаимодействия. В данный момент, не так давно это был актуальный вид чат-ботов, суть которого заключается во взаимодействии через кнопки на интерфейсе. То есть, когда пользователь впервые контактирует с ботом – его встречают приветствием, а после, предложением нажать на одну из кнопок, чтобы развивать диалог и помочь клиенту достичь необходимой цели. Это напоминает работу ботов в сфере технической поддержки в телефонии, когда вам озвучивают, какие клавиши необходимы нажать на телефоне, чтобы воспроизвести то или иное действие. Такой формат взаимодействия используется ограниченными чат-ботами.

Однако в данный момент самыми популярными являются текстовые чат-боты с интегрированными ИИ-ассистентами. Они построенны на базе нейронной сети. Такой чат-бот отслеживает слова пользователя, которые часто используются, уточняет, что пользователь имеет в виду и затем представляет варианты решения той или ной проблемы.

Различают диалоговые системы и по функциональности. Чат-боты, задача которых поддерживать диалог с клиентом, передавать ему те или иные данные, файлы, сообщения. Функциональные боты, главная цель которых предоставить пользователю возможность совершить те или иные действия, как пример – провести денежную транзакцию, пройти тест и тд. Такой вид ботов называется коммуникационным.

Диалоговые системы, основная задача которых является быть личным помощником пользователя, помогать выполнять им несложные задачи, такие как: включение музыки, получить прогноз погоды, завести будильник, позвонить и тому подобные называются ботами-ассистентами. Яркими примерами таких ботов являются “Алиса” от Yandex, “Маруся” от VK. Такие боты, в том числе популярны и в сфере образования, в сфере коммерции и развлечений.

В настоящий момент, все больше и больше современных компаний разрабатывает или заказывает себе чат-бота для технической поддержки. В большинстве случаев, пользователю можно помочь, не контактируя с ним. Это достигается написанием грамотного скрипта, который поможет пользователю в любое время понять, как решить ту или иную проблему. Если же, ситуация не описана в инструкциях – только в этом случае придется связываться с пользователем лично. Данные диалоговые системы чаще всего используются в call-центрах, в государственных учреждениях [8].

Кроме вышеуказанных диалоговых систем в основном из сфер коммерции, образования, и прочих - чат-боты также используются и в сфере

развлечений. На их базе разрабатывают мини-игры, организовывают чаты для знакомств.

2.3 Анализ и структура бота

Интеллектуальная диалоговая система вуза представляет собой программное обеспечение, имитирующее диалог с конечным пользователем - студентом, абитуриентом.

Архитектура интеллектуальной диалоговой системы вуза состоит из нескольких основных составляющих частей:

- клиентская часть,
- серверная часть,
- база данных.

Часть приложения, целью которого является взаимодействия пользователя с самим программным обеспечением, путем контакта с видимой для пользователя части называется клиентской частью [8]. Абитуриент или студент, являющийся пользователем интеллектуальной диалоговой системы, взаимодействует исключительно с этой частью и не имеет доступа и возможности взаимодействовать с серверной частью и базой данных. Чтобы пользователь мог интуитивно обращаться с чат-ботом, было принято решение использовать функциональные кнопки, по нажатию на которые интеллектуальная диалоговая система выдают ту или иную информацию, а также поле для ввода собственных вопросов. На вопросы, которые пользователь вводит самостоятельно сможет ответить ИИ-ассистент. Помимо коммуникационной задачи, клиентская часть несет ответственность за проверку данных, которые пользователь подает чат-боту. Без этой проверки есть риск некорректной работы серверной части программного обеспечения.

Часть, которая отвечает за интеграцию с ИИ-ассистентом, получение и обработку сообщений, взаимодействие с почтовыми протоколами, а также за работу с СУБД называется серверной частью.

База данных – это структурированный набор информации или данных представляем в текстовом, файловом формате, которые хранятся на основе определенных правил, хранящихся в электронном виде.

Система управления базами данных – это ПО, которое отвечает за выполнение следующих функций, связанных с базами данных: создание базы данных, поддержка в актуальном состоянии базы данных, удаление неактуальных данных, организация поиска данных в БД, а также осуществление определенного уровня доступа. После того как серверная часть принимает определенные данные от клиентской части, посылается SQL-запрос системы управления базами данных, для выполнения необходимой функции с этими данными, а затем происходит возврат запроса на клиентскую часть.

Структура взаимодействия СУБД, клиентской, серверной частей представлена на рисунке 1.

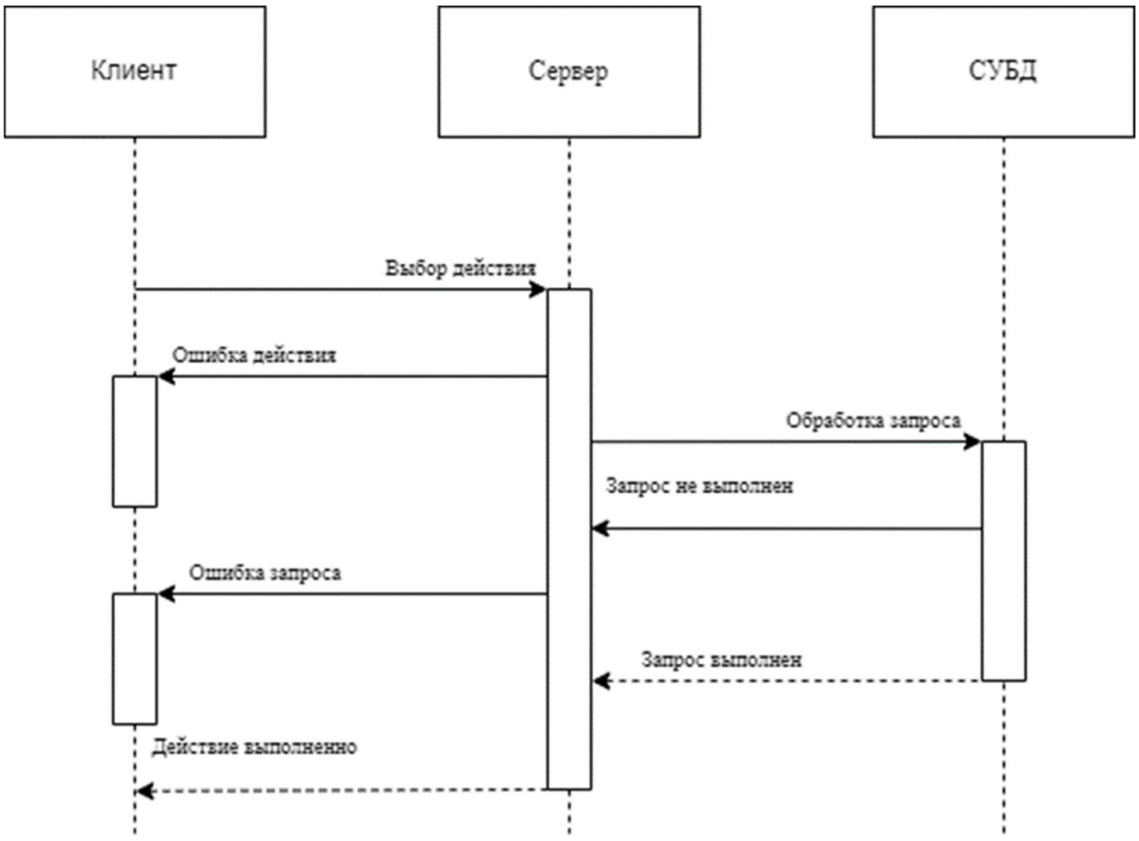


Рисунок 1 – UML-диаграмма последовательности

Для представления и отображения всех функциональных возможностей интеллектуальной диалоговой системы, необходимо создать UML-диаграмму вариантов использования, представленную на рисунке 2.

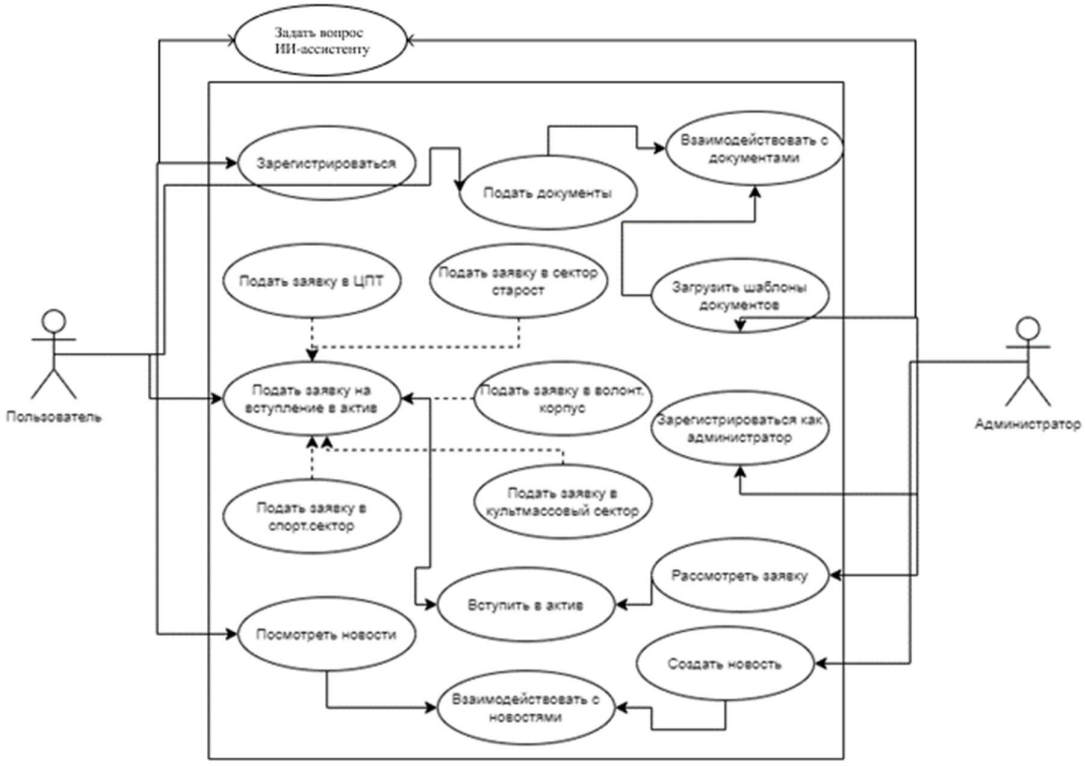


Рисунок 2 – UML-диаграмма вариантов использования

Кроме того, на рисунке 3 представлена диаграмма классов чат-бота.

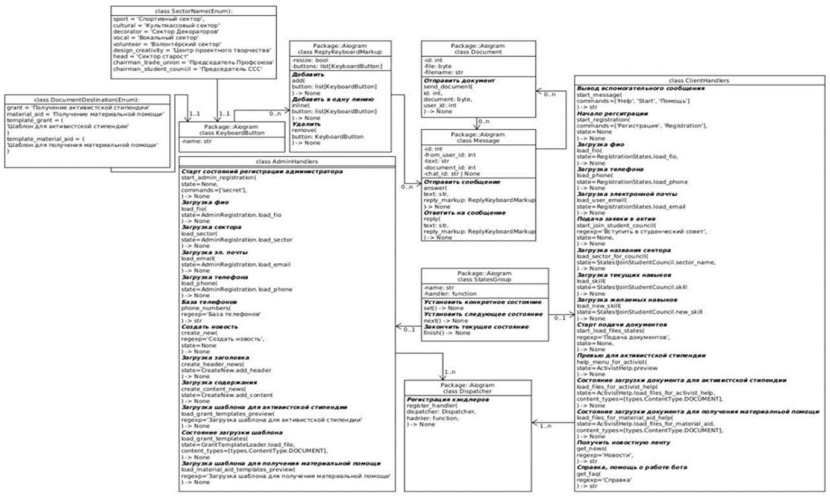


Рисунок 3 – Диаграмма классов чат-бота

2.4 Анализ видов нейронных сетей

Чтобы понимать, какого выбрать ИИ-ассистента для интеграции интеллектуальной диалоговой системой, необходимо четко понимать какие существуют виды нейронных сетей, их отличия, плюсы и минусы.

2.4.1 Полносвязные нейронные сети.

Начать анализ видов нейронных сетей стоит начать с самого базового типа искусственных нейронных сетей, а именно с полносвязных нейронных сетей. Их особенность заключается в том, что они включают в себя множество слоев нейронов, где каждый из них соединен со всеми другими нейронами как последующего, так и предыдущего слоя, за счет чего образуется плотная сеть связей.

Алгоритм полносвязных нейронных сетей представлен на рисунке 4.

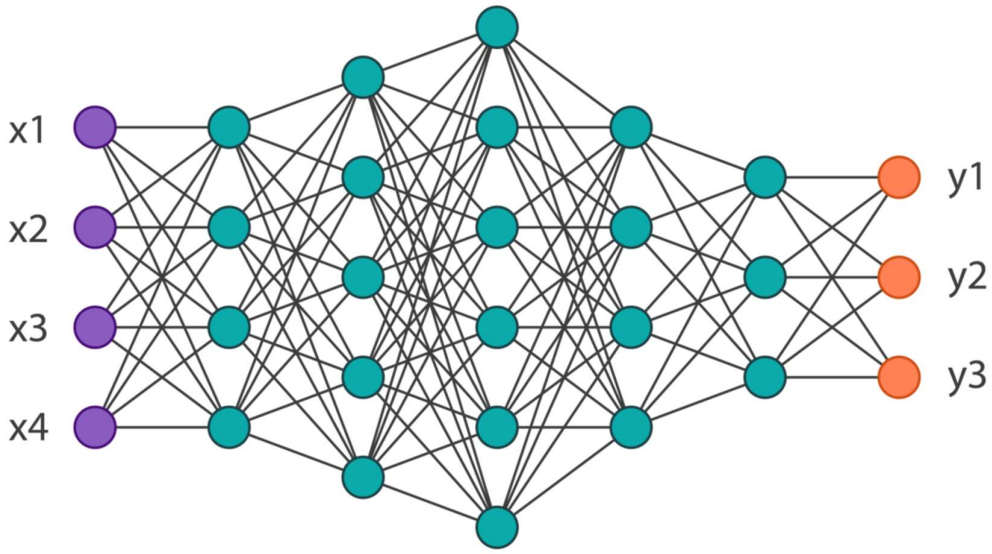


Рисунок 4 – Алгоритм полносвязных нейронных сетей

Всего различают следующие виды слоев, принимающие участие в организации стандартной нейронной сети:

– входной слой,
– скрытые слои,
– выходной слой.

Нейроны — это базовые единицы вычисления, принимающие взвешенные сигналы от предыдущего слоя, после чего суммирует их. Затем применяется функция активации и результат передается в следующий слой.

Говоря об обеспечении нелинейности в обработке данных в рамках полносвязных нейронных сетей, необходимо упоминать о такой функции активации, как сигмоид, а также о гиперболическом тангенсе и тд. Благодаря им нейросеть имеет возможность обучаться и выполнять сложные аппроксимированные функции.

Обучение полносвязных нейронных сетей построено на методе обратного распространения ошибки, также используя оптимизационные алгоритмы, чаще всего используется градиентный спуск. Обучаясь, сеть минимизирует функцию потерь, суть которой заключается в измерении рассхождения между предсказаниями сети и настоящими данными.

Данный вид сетей востребован для структурированной работы с данными, получая информацию как фиксированные наборы признаков, либо в качестве последних слоев в более сложных сетях.

2.4.2 Сверточные нейронные сети

Сверточные нейронные сети - архитектура искусственных нейросетей, которые впервые предложил французский ученый Ян Лекун в 1988 году. Цель данной архитектуры - эффективное распознавание образов, на базе которого были достигнуты прорывы в области компьютерного зрения и обработки изображений.

Выделяют следующие виды слоев, принимающие участие в организации сверточной нейронной сети:

– сверточный слой - слой, отвечающий за обнаружение локальных паттернов внутри данных, например: края, углы, текстуры,

– пулинговый слой - слой, отвечающий за уменьшение размерности карт признаков, а также за увеличение их инвариантности к масштабированию

и небольшим искажениям. Достигается этот процесс самым распространненым типом пулинга – максимальным,

– полносвязный слой - слой, отвечающий за итоговую “сборку” и классификацию изображений по категориям.

Обучение сверточных нейронных сетей также, как и у полносвязных нейронных сетей построено на базе метода обратного распространения ошибки и оптимизационных алгоритмов, к примеру - стохастический градиентынй спуск. Кроме того, часто используются различные методы аугментации данных, например: вращение, отражение изображений, масштабирование и тд.

Сверточные нейронные сети чаще всего используются в сферах медицины, безопасности и прочих. Благодаря исследованиям данного вида сетей, в современности мы можем пользоваться распознаванием лиц, детекцией объектов, сегментацией изображений и прочими достижениями науки.

2.4.3 Рекуррентные нейронные сети

Рекуррентные нейронные сети - сети, цель которых заключается в обработке последовательных данных, где порядок и связность элементов имеют большое значение. Яркой отличительной чертой подобных нейронных сетей заключается в том, что они учитывают текущее входное значение, а также хранят информацию о предыдущих состояниях.

Рекуррентные нейронные сети имеют обратные связи, позволяющие передавать информацию от выходов предыдущих шагов к входам, текущим шагов, за счет чего создается внутренняя память сети, которая используется для сохранения контекста во время обработки последовательностей данных.

Алгоритм рекуррентных сетей представлен на рисунке 5.

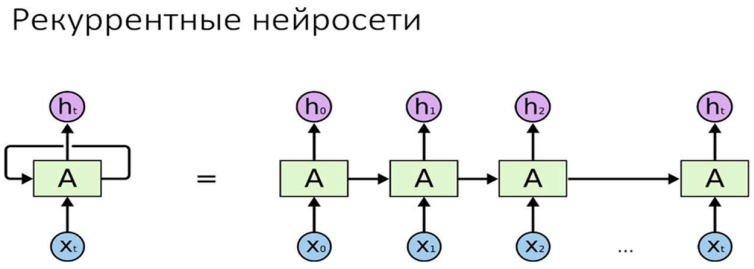


Рисунок 5 – Рекуррентные нейросети

Минусами в обучении рекуррентных нейронных сетей являются затухание и взрывающиеся градиенты, который затрудняют обучение на длинных последовательностях данных. Данный вид сетей повсеместно используется для задач машинного перевода, для создания виртуальных помощников, а также для обработки речи.

2.4.4 Сети прямого распространения

Сети прямого распространения - сети, где связь между нейронами проходит лишь в единственном направлении, от входного слоя к выходному слою последовательно. Из-за подобной структуры организации нейронов в данном виде нейронных сетей отсутствуют обратные связи и циклы.

Главным плюсом сетей прямого распространения является их способность к аппроксимации любых функций. Данное открытие совершил Джорж Цыбенко в 1989 году. Его теория гласит, что искусственная нейронная сеть прямой связи с одним скрытым слоем может аппроксимировать любую непрерывную функцию многих переменных с любой точностью.

Сети прямого распространения используются в задачах классификации и регрессии. Они состоят из следующих видов слоев:

– входной слой, принимающий данные на входе,

– скрытые слои, которые обрабатывают данные, полученные на входном слое через весовые связи и функции активации,

– выходной слой, который является завершающим звеном в системе слоев. Данный вид слоя предоставляет конечный результат работы нейронной сети.

На рисунке 6 представлена организация слоев сетей прямого распространения.

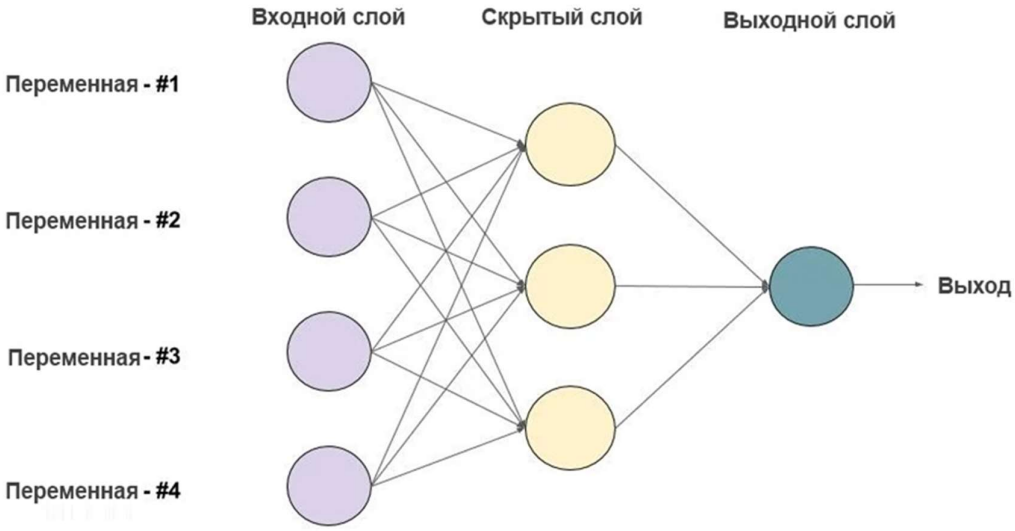


Рисунок 6 – Слои сетей прямого распространения

Как и многие другие виды нейронных сетей, сети прямого распространения также обучаются на основе алгоритма обратного распространения ошибки, благодаря которому появляется возможность корректировки веса соединений между нейронами на основе разницы между фактическими выходными данными и предсказанными.

Данный вид сетей в основном используется в сферах, для которых важен анализ данных, а также принятие решений. Сети прямого распространения используются для распознавания речи и изображений, финансовых прогнозов и тд.

2.4.5 Генеративно-состязательные сети

Генеративно-состязательные сети - вид нейронных сетей, основная особенность которых - отсутствие “учителя”, а обучение нейросети построено на основе синтеза двух нейронных сетей, задачей одной из них является генерация образцов, а цель другой - разбор верных и ложных образцов. Родоначальником данного вида нейронных сетей является Иэн Гудфеллоу в 2014 году, предложивший соответствующую концепцию.

На рисунке 7 изображен алгоритм генеративно-состязательных сетей.

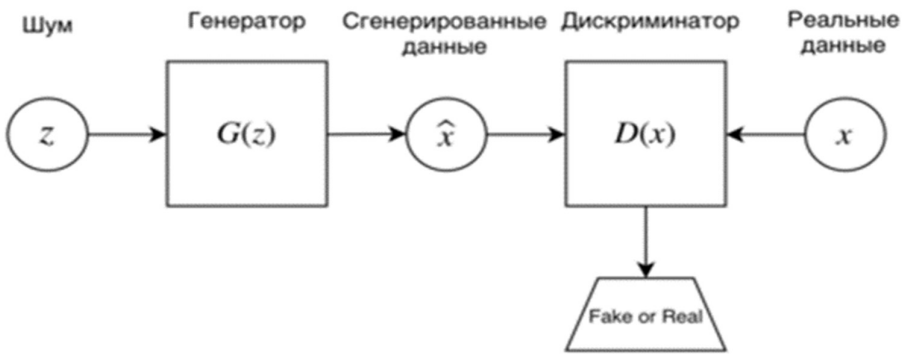


Рисунок 7 – Генеративно-состязательные сети

Процесс обучения ГСС итеративен. Генератор получает случайный шум и преобразует его в изображение. Дискриминатор принимает как реальные изображения из тренировочного набора данных, так и синтетические изображения, созданные генератором, и обучается классифицировать их как «настоящие» или «поддельные». Генератор, в свою очередь, обновляется в направлении повышения своей способности вводить в заблуждение дискриминатор.

Главным преимуществом генеративно-состязательных сетей является генерация высококачественных, практически “реальных” данных. Данный вид сетей используется во многих приложениях для созданий фотографий, которых на самом деле не существует. Развитие данного вида нейронных сетей привело к созданию такого понятия, как “deepfake” - синтез изображения, с помощью которого можно “подделывать” изображения, видео и голос.

2.4.6 Автоэнкодеры

Автоэнкодер - тип искусственных нейронных сетей, целью которых является нелинейное снижение размерности. Автоэнкодеры состоят из двух сущностей: кодировщик и декодировщик.

Задачей кодировщика является сжатие входных данных в скрытый слой, а декодирвщик восстанавливает исходные данные.

Виды автоэнкодеров:

– базовый автокодировщик, который состоит из трех слоев – входного, одного скрытого, и выходного,

– сжимающий автокодировщик, использующий регуляризацию в виде редкости на скрытом слое, что заставляет модель изучать более информативные признаки,

– шумоподавляющий автокодировщик, который обучается на очистке данных от добавленного шума,

– вариационный автоэнкодер, суть которого заключается в определении параметров распределения вероятностей для каждого измерения внутреннего представления,

– сверточный автоэнкодер, который использует свёрточные слои вместо полносвязных, что позволяет эффективнее работать с изображениями.

Основными задачи автокодировщиков в наше время являются задачи по уменьшению шума на изображениях, а также по снижению размерности.

2.5 Постановка задачи

Необходимо спроектировать и разработать интеллектуальную диалоговую систему вуза на основе telegram для помощи абитуриентам в выборе института на основе мирового рейтинга, а также для помощи действующим студентам в учебной программе путем внедрения ИИ-ассистента. Кроме того, необходимо учесть запрос студентов, состоящих в советах студенческих самоуправлений, на оптимизацию подачи заявлений на получение материальной помощи и прочих документов, а также на упрощение процесса вступление в совет студенческого самоуправления. Чат-бот должен предоставлять возможность со стороны пользователя: подать заявку на вступление в ССС, возможность просмотра новостной ленты, возможность получения шаблона, а также отправки заполненного шаблона на почту деканата, вызов ИИ-ассистента как на основе загружаемого файла, так и на основе собственной базы данных. Со стороны администратора: загрузка шаблона документов, создание новостных постов, просмотр базы заявок, вызов

ИИ-ассистента как на основе загружаемого файла, так и на основе собственной базы данных.

2.6 Описание архитектуры проекта

Поставив цели разрабатываемого проекта, было принято решение построить архитектуру интеллектуальной диалоговой системы вуза, изображенную на рисунке 8.

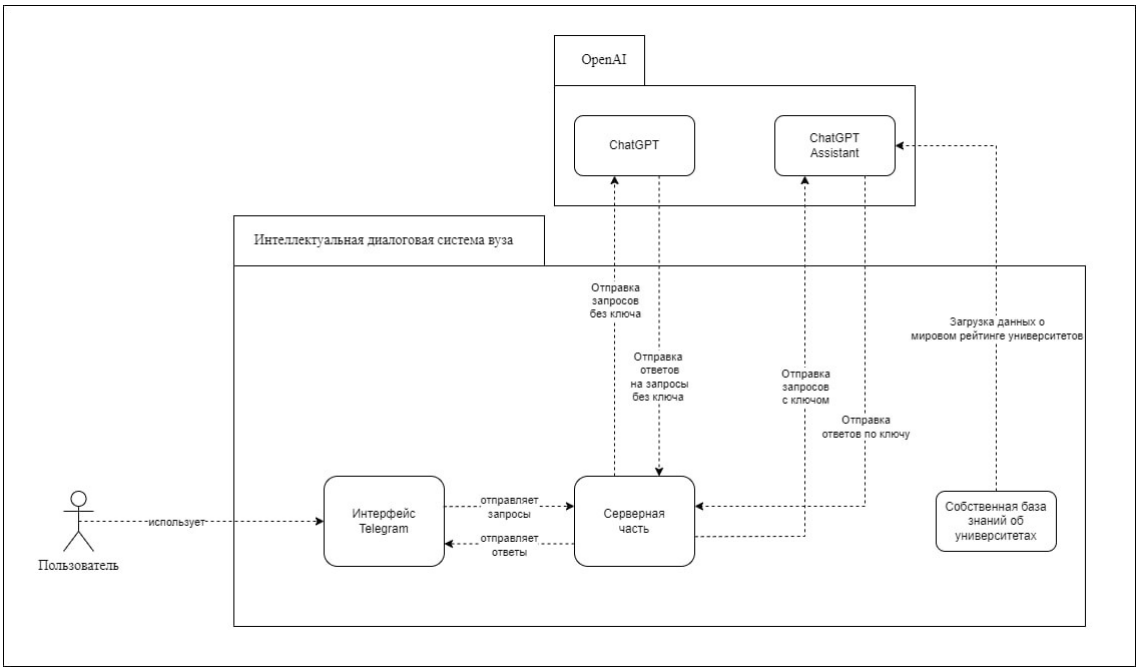


Рисунок 8 – Архитектура интеллектуальной диалоговой системы вуза

В процессе работы с контентом пользователь взаимодействует с интерфейсом мессенджера Telegram, на котором базируется искусственная диалоговая система вуза.

Интерфейс состоит из кнопок, по нажатию на которые выполняются те или иные действия системой и поля для ввода собственных сообщений пользователем.

Серверная часть после принятия сообщения от пользователя проводит проверку, в каком формате пришел запрос. Система может принять входящие от пользователя сообщения в следующих видах:

– нажатие на кнопку,

– текстовый запрос с ключом,

– текстовый запрос.

Если пользователь не вводит сообщения вручную, а нажимает на какую-либо кнопку – в таком случае подключение к ChatGPT не производится, т.к. чат-бот выполняет строго прописанные для него инструкции при нажатии на эти кнопки.

Если же пользователь вручную вводит какой-либо запрос, например, “объясни мне третий закон Ньютона”, то в таком случае система производит проверку на наличие специального ключа.

Если в запросе отсутствует специальный ключ, то система отправляет текст сообщения пользователя на сервер OpenAI, где ответ генерируется моделью ChatGPT, после чего также отправляется на серверную часть системы, а затем отправляется на интерфейс telegram-бота.

Если в запросе фигурирует специальный ключ, который зашит в коде системы, в нашем случае psu, то в таком случае создается отдельный поток, внутри которого создается ассистент со стороны OpenAI через подключение к Assistant API, который выдает ответы исключительно по файлу, который загружается на сервер OpenAI системой.

Независимо от вида запроса, интеллектуальная диалоговая система вуза возвращает пользователю текстовый ответ на поставленный пользователем вопрос.

2.7 Выбор инструментов разработки

2.7.1 Выбор мессенджера

Изучая мессенджеры, на которых можно базировать своего чат-бота, были обнаружены следующие популярные на территории СНГ приложения: WhatsUp, Viber, Telegram.

Telegram – это кроссплатформенный мессенджер, самый популярный и безопасный мессенджер на сегодняшний день. Сервера Telegram распределены по всему миру для обеспечения быстродействия и защищенности. Мессенджер

отличается бесплатным распространением и полным отсутствием платных подписок и рекламы, также предоставляй открытые API для разработчиков. Самое главное, мессенджер прост и удобен в использовании, обладает наиболее интуитивно понятным интерфейсом, в сравнении с его основными конкурентами, например, WhatsApp и Viber. Выбор пал на Telegram, за счет плюсов, указанных выше. Кроме того, благодаря специальному telegram bot API – реализация чат-бота становится более тривиальной.

2.7.2 Выбор языка программирования

Занимаясь разработкой такого программного продукта, как чат-бот – необходимо ответственно подойти к выбору инструментов разработки, а именно: мессенджер, язык программирования, СУБД.

Анализируя разработку чат-ботов, следующим важнейшим инструментом является ЯП. Данный программный продукт можно реализовать на таких языках, как: PHP, JavaScript, C#, GO, Python и т.д.

Java – это язык программирования общего назначения, разрабатываемый компанией Oracle, относящийся к языкам программирования с сильной типизацией и объектно-ориентированным языкам. Важной особенностью языка Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы, например, попытка несанкционированного доступа к данным или соединения с другим компьютером, вызывают немедленное прерывание [1].

Python – это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Python является открытым программным обеспечением, то есть можно свободно распространять копии этого программного обеспечения, читать его исходные тексты, вносить изменения, благодаря этому, Python был портирован на много платформ (т.е. изменён таким образом, чтобы работать на них). Все ваши программы смогут запускаться на

любой из этих платформ без каких-либо изменений, если только вы избегали использования системно-зависимых функций [2]. С каждым годом язык лишь набирает популярность среди разработчиков. Кроме того, благодаря специальной асинхронной библиотеке aiogram – разработка ботов становиться тривиальной.

C# – это язык программирования с Си-подобным синтаксисом. C# на протяжении долгого времени уверенно держит позиции в рейтинге наиболее востребованных на рынке разработки языков. C# отличается очень богатым синтаксисом, который в то же время является простым и удобным. Одним из основных достоинств C# является наличие мощных функций, и предоставление прямого доступа к операционной памяти [3]. Недостатком языка является приоритетная ориентированность на платформу Windows.

2.7.3 Выбор СУБД

Для проектирования базы данных чат-бота необходимо выбрать систему управления базами данных. Для решения поставленной задачи, необходимо выбрать такую СУБД, которая будет проста в настройке и развертывании, отказоустойчива, безопасна и стабильна.

PostgreSQL – масштабируемая объектно-реляционная СУБД, что даёт ей некоторые преимущества над другими SQL базами данных с открытым исходным кодом, такими как MySQL, MariaDB и Firebird. PostgreSQL работает на Linux, Windows, OSX и некоторых других системах.

Фундаментальная характеристика объектно-реляционной базы данных — это поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. Это делает PostgreSQL невероятно гибким и надежным. Среди прочего, он умеет создавать, хранить и извлекать сложные структуры данных. В данной СУБД встречаются вложенные и составные конструкции, которые не поддерживаются стандартными реляционными системами управления базами данных [4].

Oracle RDBMS (она же Oracle Database) на первом месте среди СУБД. Система популярна у разработчиков, проста в использовании, у нее понятная документация, поддержка длинных наименований, JSON, улучшенный тег списка и Oracle Cloud. О том, что Oracle лидер среди других СУБД говорит хотя бы тот факт, что по данным полученным в 2007 году, СУБД охватывала 47% мирового рынка этой отрасли [5].

Однако, данная СУБД имеет и ряд своих недостатков:

– Oracle для частного или малого использования предоставляет бесплатную лицензию, но более крупных пользователей покупка лицензии крайне затратно,

– настройка и развертывание Oracle крайне трудоемко и зачастую требует квалифицированной поддержки,

– Oracle значительно менее популярна в Open Source коде, где большинство разработчиков используют MySQL и PostgreSQL,

– система может потребовать значительных ресурсов уже сразу после установки, поэтому возможно потребуется модернизировать оборудование для внедрения Oracle.

Полная база данных SQL содержится в одном файле на диске, и все операции чтения и записи выполняются непосредственно в этом файле на диске. Поскольку данные записываются непосредственно обратно в файл диска, SQLite придерживается атрибута ACID для защиты транзакций от сбоев выделения памяти и ошибок ввода-вывода диска, вызванных непредвиденными сбоями системы или сбоями питания.

Библиотека SQLite является одной из самых компактных библиотек в этом списке, где размер библиотеки может быть меньше 600 КБ.

Некоторые другие хорошие примеры использования включают веб-сайты с низким и средним трафиком (около 100 тыс. Запросов в день), тестирование и внутреннюю разработку, использование Tcl или Python для анализа данных и в

образовательных целях (это легко настроить и может быть использовано для обучения SQL концепциям учащихся) [6].

2.7.4 Выбор готового ИИ-ассистента

Chatsonic - нейросеть, основная задача которой в создании различных текстовых данных. Нейросеть представляет собой интерфейс диалоговой системы на основе чат-бота и пользуется популярностью у представителей профессий, работающих с контентом для социальных сетей, новостных порталов, блогов и тд.

У данной нейросети есть три основные функции:

1) краткие ответы на вопросы в текстовом формате. История сообщений запоминается ботом, а свои ответы он генерирует с помощью поисковой системы Google,
2) Генерация изображений по запросу,
3) Генерация профессионального контента.

Используя промпты, вы можете “задать” чат-боту то или иное поведение. Вы можете “попросить” бота перенять поведение поэта, известного разработчика или ученого.

Минусами ChatSonic является - зависимость от данных, ошибочные ответы по этических и моральным вопросам, ограниченная креативность.

Плюсом ChatSonic является поддержка русского языка, наличие бесплатной ограниченной версии, наличие разработанного API для интеграций.

ChatGPT от OpenAI представляет собой диалоговую систему с генеративным искусственным интеллектом. Чат-бот отвечает на различные вопросы, генерирует ответы различных форматов. Данная модель обучается методом обучения с учителем, а также обучения с подкреплением. Последняя актуальная версия является ChatGPT 4. В новой версии диалоговой системы есть возможность обрабатывать как текстовую информацию, так и изображения.

С его помощью можно сгенерировать готовые функции кода, переводить тексты с иностранных языков, получать точные ответы на основе базы данных, а также с применением поисковой системы Google.

ChatGPT пользуется особой популярностью, благодаря чему представлен как в виде диалоговой системы на базе веб-страницы, так и имеет собственные приложения для Android и IOS.

Особенностью ChatGPT 4 является обучение ее на максимально большом массиве информации, а также огромные вычислительные мощности сервера.

Минусами ChatGPT является то, что ее использование ограниченно на территории РФ, и для работы с ней необходимо воспользоваться VPN.

Несмотря на ограниченное использование в России, полнота данных и возможность работать как с текстом, так и с изображениями делают ChatGPT одним из ведущих ИИ-ассистентов на данный момент.

YandexGPT 2 - обновленная версия новой отечественной диалогой системы, построенный на основе генеративной языковой модели. Данная система может сгенерировать любой текст и помочь в выполнении сложных задач.

Благодаря данной системе, пользователь сможет стилизовать текст под определенную аудиторию и специфику, выделять в тексте главные мысли и детали, отвечать на те или иные вопросы пользователя, создавать текст по запросу с нуля, а также генерировать названия, слоганы, писать сценарии и многое другое.

Система интегрирована в умную колонку от Яндекса “Алису”, в Яндекс.Браузер, что помогает использовать модель прямо во время пользования поиска информации в интернете.

Минусами YandexGPT 2 являются:

– Возможная недостоверность информации за счет подражания текстом из интернета,

– При некорректных вопросам с яркой негативной и обидной окраской - чат-бот может вообще не выдать ответ на вопрос,

– База знаний YandexGPT 2 составлена лишь до марта 2023 года, а это значит, что диалоговая система заранее устаревшая и выдает неактуальные данные.

К плюсам можно отнести следующие факты:

– наличие более продвинутой версии YandexGPT 3,

– наличие API для интеграций,

– полностью отечественная модель.

2.8 Выбранные технологии

В ходе проведенных анализов, было приняты следующие решения:

– выбрать Telegram как мессенджер для базирования чат-бота,

– выбрать среду разработки Pycharm IDE и язык программирования python с последующим применением библиотеки sqlalchemy и aiogram,

– выбрать PostgreSQL для управления базами данных,

– выбрать ChatPGT 4 для выполнения функций ИИ-ассистента.

3 Реализация

3.1 Регистрация чат-бота

Перед тем, как начать разрабатывать диалоговую систему и писать код, необходимо зарегистрировать нового чат-бота в Telegram. Для этого необходимо зарегистрироваться в мессенджере, и затем найти бота “@BotFather”, представленного на рисунке 9.

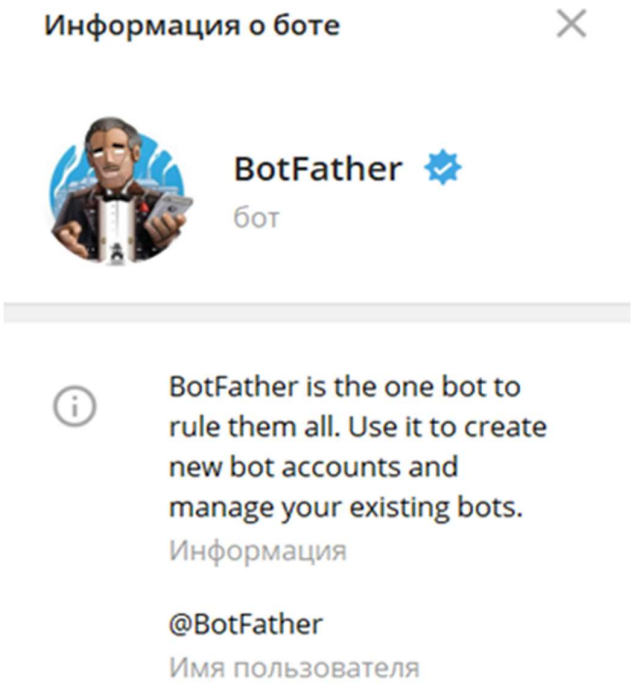


Рисунок 9 – BotFather

После того, как чат с ботом начат, необходимо запустить бота командой “/start”, представленной на рисунке 10.

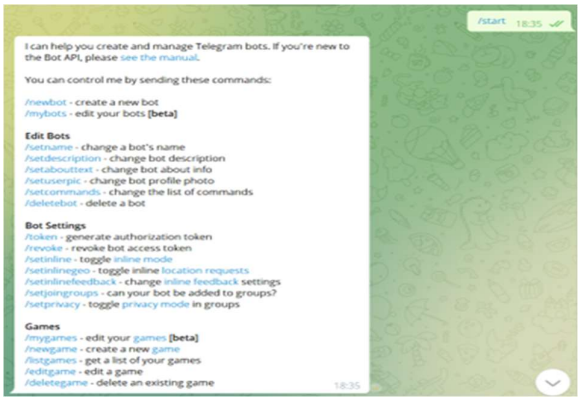


Рисунок 10 – Команда /start

Чтобы создать бота непосредственно для нашей задачи, необходимо ввести команду /newbot, а затем дать боту имя, и логин. Процесс создания бота представлен на рисунке 11.

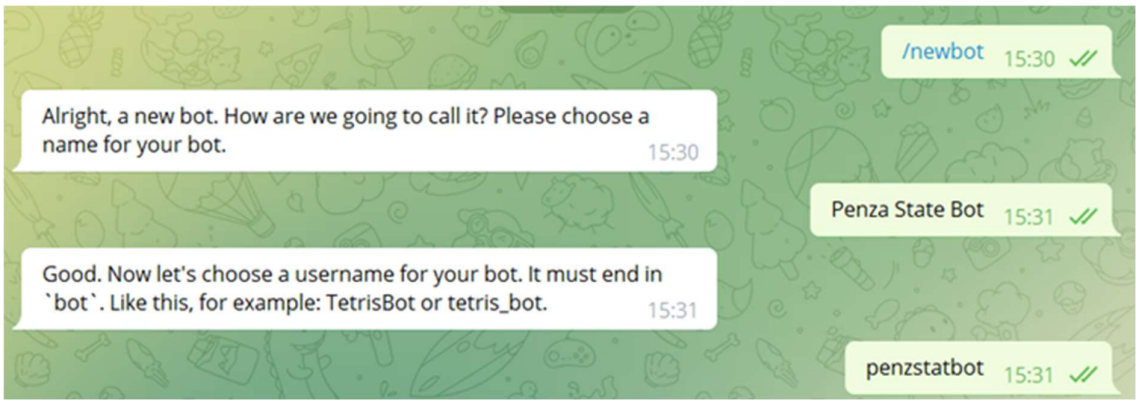


Рисунок 11 – Создание бота

Далее приходит сообщением с токеном бота, необходимым для связи Telegram и нашего бота. Полученное сообщение представлено на рисунке 12.

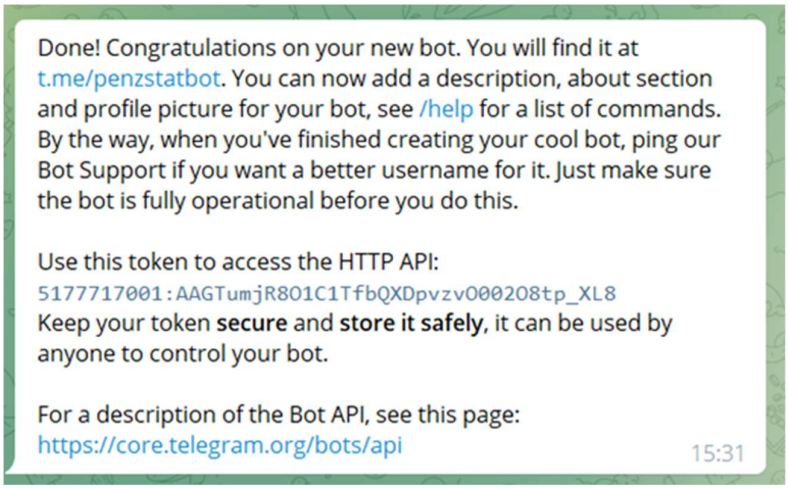


Рисунок 12 – Токен бота

3.2 Работа с БД

Для хранения информации необходимо реализовать базу данных. Таблица user необходима для содержания информации о пользователях бота. Поля данной таблицы указаны в таблице 1.

Таблица 1 – Атрибуты таблицы “user”

Имя столбца	Тип данных	Описание	Роль атирбута	Пример
id	integer	Идентификатор пользователя	PK	1
fio	varchar	Фамилия, имя, отчество пользователя		Сидоров Евгений Андреевич
tg_user_id	integer	Идентификатор пользователя в телеграм		112631
phone	varchar	Телефонный номер пользователя		89270992634
email_address	varchar	Адрес электронной почты пользователя		example@mail.ru
status	varchar	Роль в системе		Пользователь
sector_name	varchar	Наименование сектора, за которым закреплён администратор		Сектор старост

Таблица document предназначена для содержания информации о загруженных документах и их назначении. Поля данной таблицы указаны в таблице 2.

Таблица 2 – Атрибуты таблицы “document”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	integer	Идентификатор документа	PK	1
document	byte	Закодированный файл		0x255044D16D14144D14E350A321
file_name	varchar	Имя файла		Шаблон
owner	integer	Владелец документа	FK	1
destination	varchar	Назначение документа		Получение материальной помощи
created	datetime	Дата загрузки		20220605 18:21

Таблица student_council_application предназначена для хранения информации об отправленных заявках в актив. Поля данной таблицы указаны в таблице 3.

Таблица 3 – Атрибуты таблицы “student_council_application”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	integer	Идентификатор заявки	PK	3
sector_name	varchar	Название сектора		Сектор старост
skill	varchar	Текущие навыки		Умею хорошо петь
new_skill	varchar	Желаемые навыки		Хочу научиться работать в команде
user_id	integer	Идентификатор пользователя	FK	23
created	datetime	Дата создания заявки		20220602 18:26

Таблица news предназначена для хранения информации о новостях университета. Поля данной таблицы указаны в таблице 4.

Таблица 4 - Атрибуты таблицы “news”

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	integer	Идентификатор новости	PK	2
creator	integer	Создатель новости	FK	21
header	varchar	Заголовок		Заголовок
content	varchar	Содержание новости		Наш факультет победил на соревнованиях по шахматам.
created	datetime	Дата создания новости		20220602 21:22

Схема таблиц БД приведена на рисунке 13.

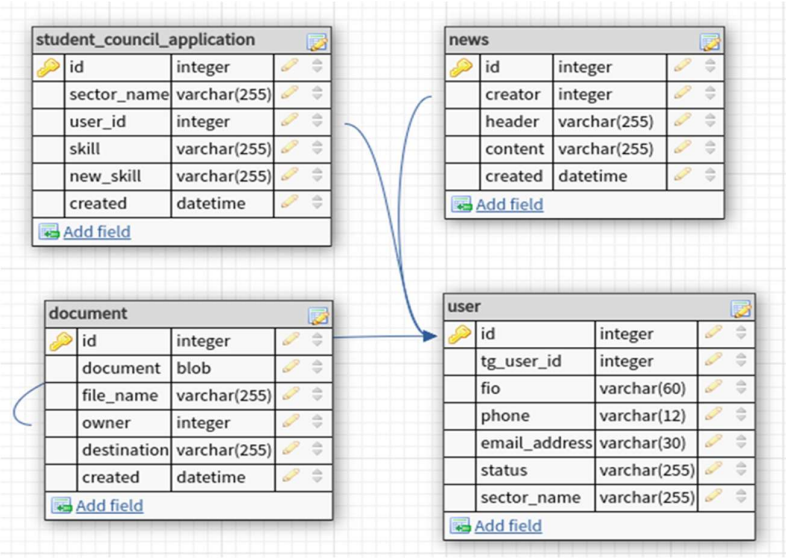


Рисунок 13 - Схема таблиц БД

3.3 Описание доступных функций

Приложение предполагает реализацию таких функций, как:

– вызов ИИ-ассистента,

– подача заявки на вступление в сектор активистов,

– почтовое оповещение руководителя сектора о поступившей заявке,

– создание новостных постов и их просмотр,

– просмотр базы номеров активистов,

– регистрация студентов в качестве активистов и в качестве администратора,

– загрузка файла-шаблона заявления на материальную помощь и на активистскую стипендию,

– отправка заполненных заявлений,

– информационный блок о секторах[8].

Чтобы запустить бота, необходимо прописать команду /start. После этого, пользователь может либо ввести команду /registration, чтобы зарегистрироваться в качестве рядового активиста-студента, либо ввести команду /Admin_reg, чтобы зарегистрироваться в качестве руководителя сектора, либо председателя факультета. У разных пользователей определен различный функционал. В данный момент, пользователю не предлагается команда /Admin_reg, так как является скрытной от обычных пользователей, не являющихся уполномоченными.

Старт работы бота представлен на рисунке 14.

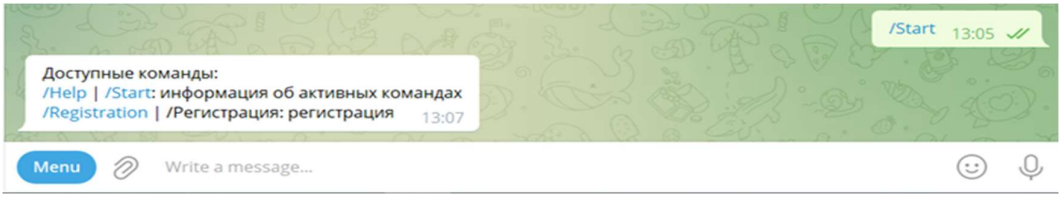


Рисунок 14 – Старт работы бота

При вводе команды регистрации, пользователю предлагается пройти небольшой опрос со следующими полями: имя и фамилия, номер телефона, электронная почта. После успешной регистрации, пользователю открывается

кнопочное меню, в которых открываются все функции, доступные пользователю. Регистрация показана на рисунке 15.

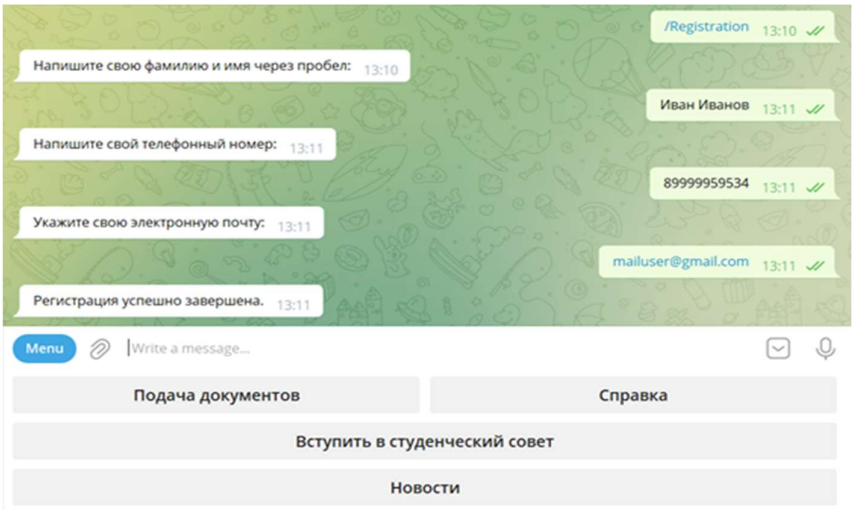


Рисунок 15 – Регистрация пользователя

Однако, пользователь может допустить ошибки при вводе данных на регистрации, тогда чат-бот выдаст следующие сообщения. Обработка ошибок представлена на рисунке 16.

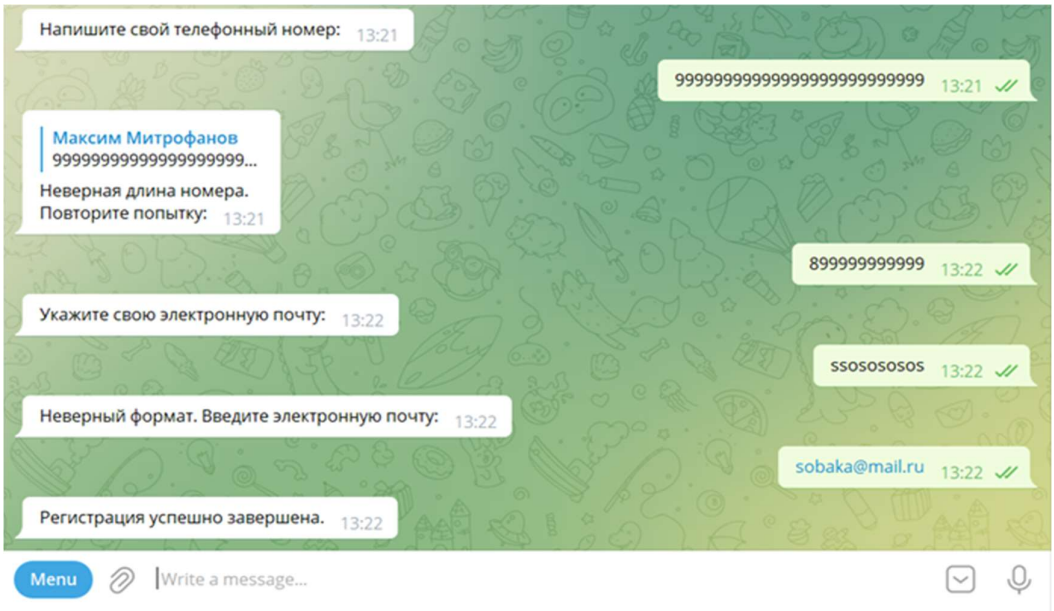


Рисунок 16 – Обработка ошибок

Регистрация администратора осуществляется похожим способом, отличие лишь в команде /Admin_reg и дополнительное поле “курируемый сектор”. После регистрации, администратору предлагается весь функционал

рядового пользователя, а также возможность просмотреть базу номеров телефона, возможность добавить новости, а также загрузить шаблон заявления на получение мат. помощи или активистской стипендии[8].

На рисунке 17 представлена регистрация пользователя с ролью администратора.

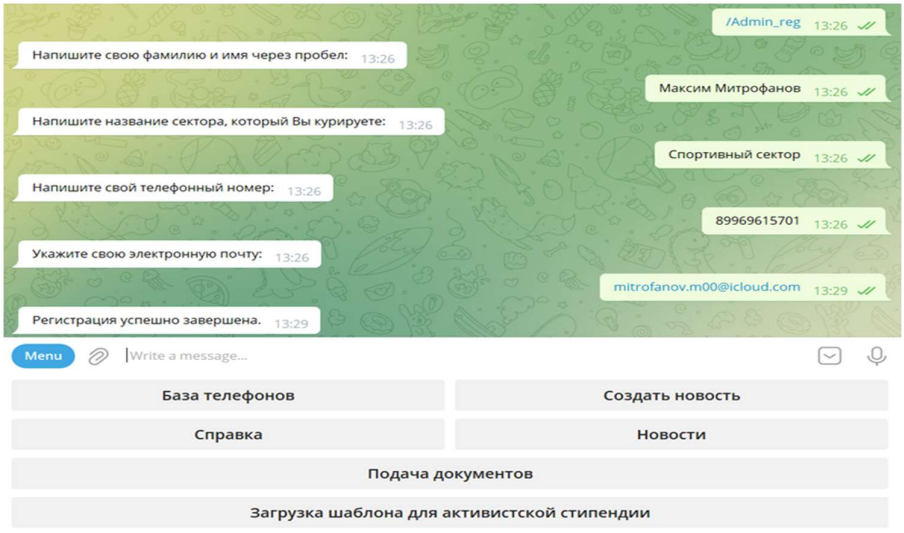


Рисунок 17 – регистрация пользователя с ролью администратора

После успешной регистрации, администратор благодаря кнопке “База телефонов” может просмотреть всех зарегистрировавшихся пользователей. На рисунке 18 представлен просмотр базы телефонных номеров[8].

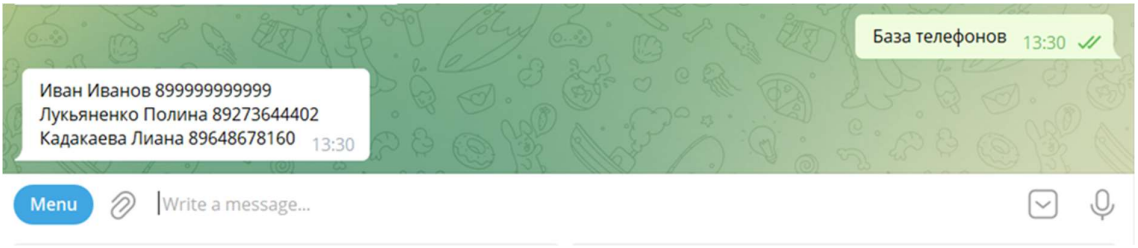


Рисунок 18 – Просмотр базы телефонных номеров

Администратор может создавать новостные записи, которые будут доступны каждому пользователю. Для создания поста, необходимо ввести заголовок новости и сам текст[8].

На рисунке 19 представлен создание новостей.

Рисунок 19 – Создание новостей

После создания новостного поста, пользователь и администратор могут просмотреть новостную ленту. На рисунке 20 представлен просмотр новостей.

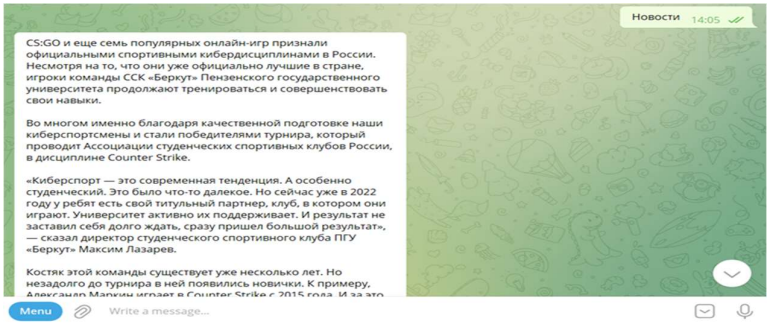


Рисунок 20 – Просмотр новостей

Кнопка “Справка” необходима для предоставления помощи пользователю о функционале данного чат-бота. На рисунке 21 представлена работа кнопки “Справка”.

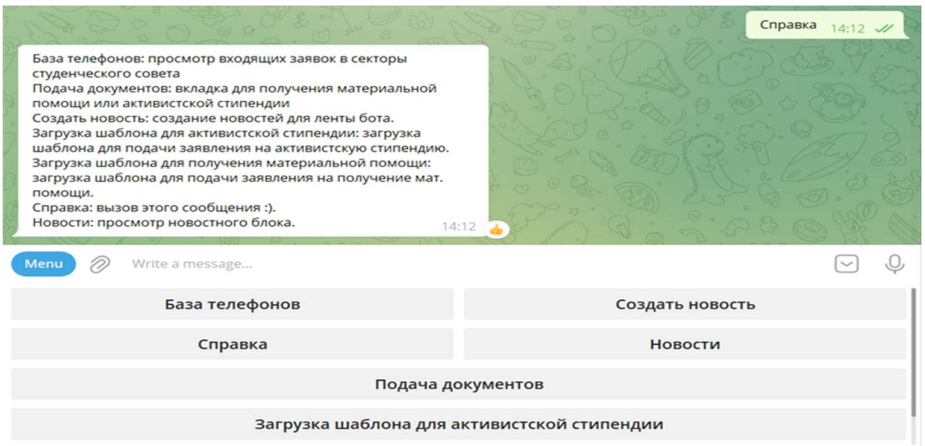


Рисунок 21 – Работа кнопки “Справка”

Одна из функций чат-бота – подача заявки на вступление в тот или иной сектор органа студенческого самоуправления. Для того, чтобы подать заявку, необходимо нажать кнопку “Вступить в студенческий совет” и заполнить следующие поля: желаемый сектор, навыки и умения, цели[8].

На рисунке 22 представлена реализация заявки на вступление в сектор.

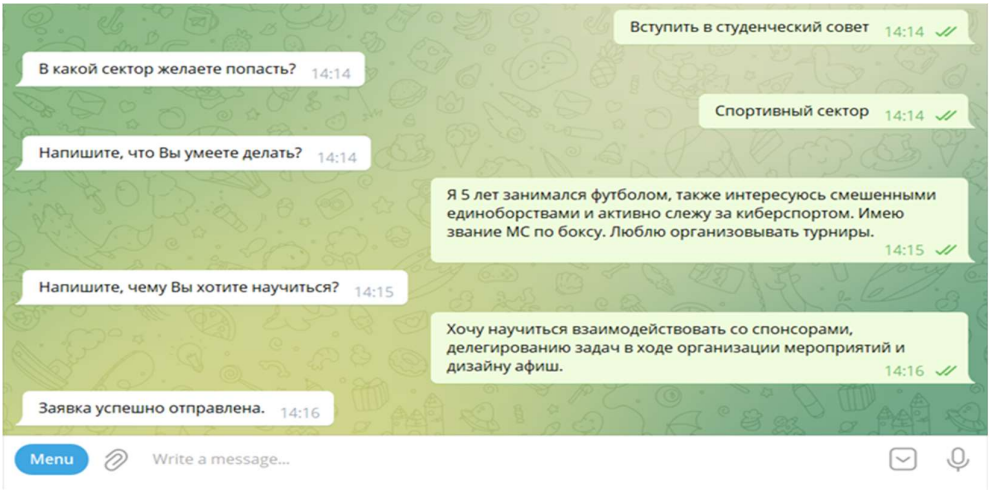


Рисунок 22 – Реализация заявки на вступление в сектор

После того, как пользователь подает заявку на вступление в тот или иной сектор, руководителю данного сектора, человеку, который зарегистрировался как администратор сектора – придет уведомление на почту. На рисунке 18 представлено email-оповещение о заявке[8].

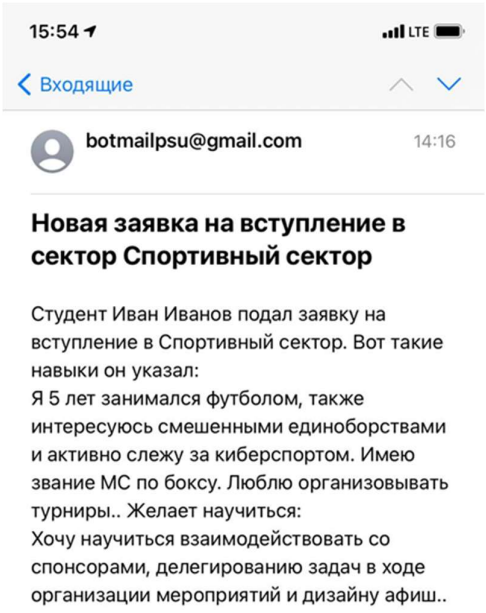


Рисунок 23 – email-оповещение о заявке

В ходе разработки находится алгоритм подачи заявления на получение материальных благ. Администратор может загружать шаблоны заявлений, в то время как другие пользователи получают их в сообщениях, и отправляют заполненные заявления боту, а тот в свою очередь – отсылает на почту председателя профсоюза[8].

На рисунке 24 представлена загрузка шаблона.

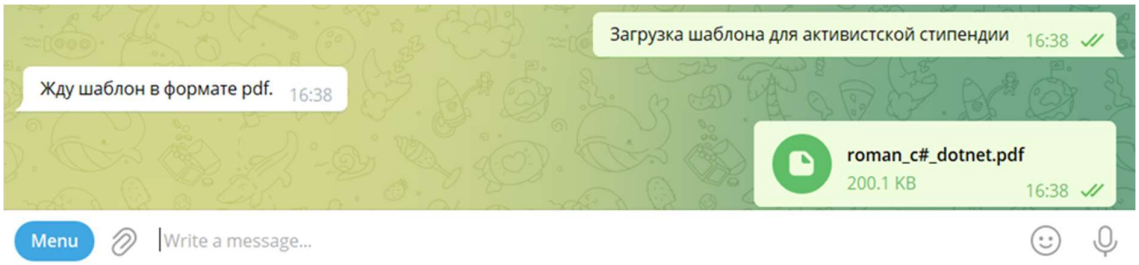


Рисунок 24 – Загрузка шаблона

С помощью интеллектуальной диалоговой системы можно вызвать ИИ-ассистента. Для того чтобы обратиться с любым вопросом к боту, достаточно просто ввести команду /start, после чего вы уже можете сразу общаться с помощником. На рисунке 25 отображен ответ ИИ-ассистента.

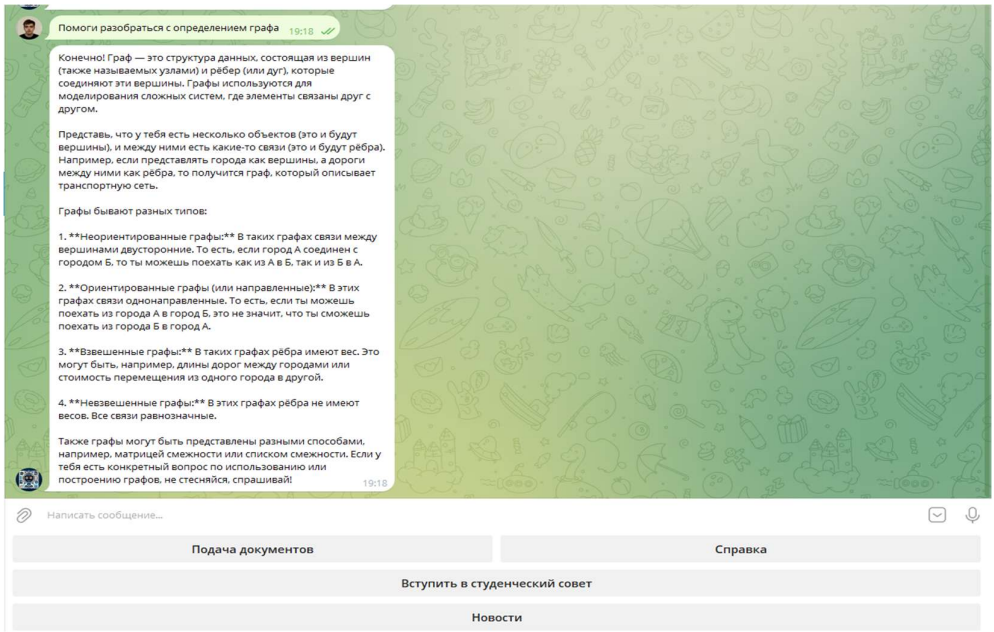


Рисунок 25 – Общение с ИИ-ассистентом

Помимо использования базы данных, которая изначально есть в версии ChatGPT, подключенной к интеллектуальной диалоговой системе, мы также

можем задать вопрос по собственному файлу, в котором собраны мировой рейтинг вузов. Для этого необходимо уточнить чат-боту, что необходимо отвечать на основе загруженного файла. На рисунке 26 отображена работа по файлу.

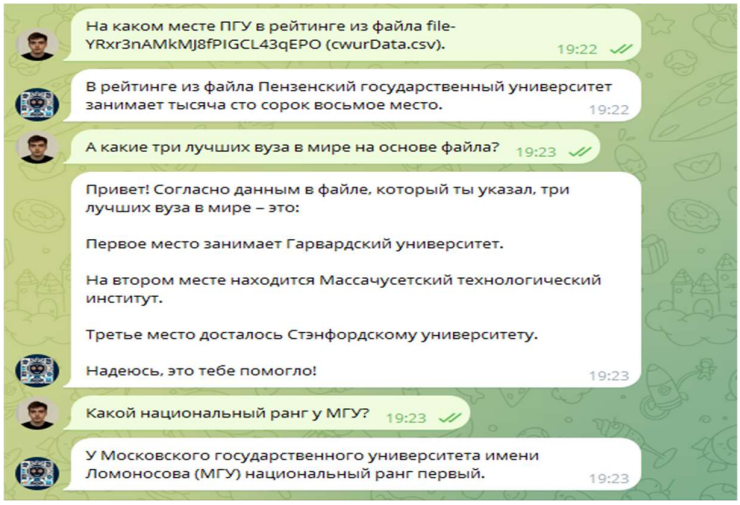


Рисунок 26 – Общение с ИИ-ассистентом на основе собственной базы знаний

3.4 Описание программы

Диаграмма развертывания, приведенная на рисунке 27, отображает все возможное физическое развертывание артефактов на узлах.

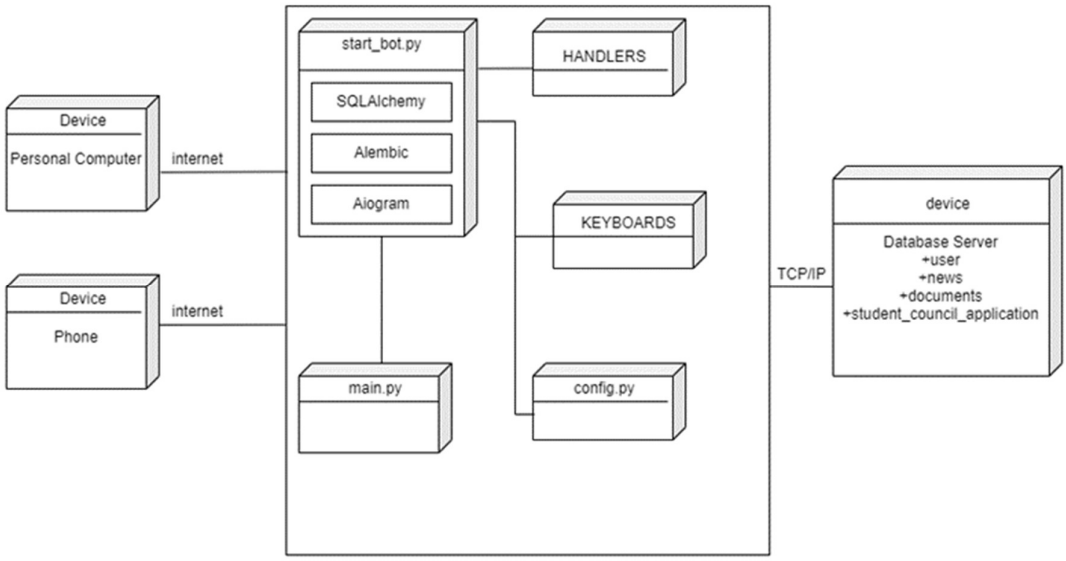


Рисунок 27 - UML-диаграмма развертывания

Файл gitignore.py - файл, который ограничивает попадание в репозиторий скомпилированных файлов питона, различных кэшей и локальной БД.

Файл config.py - файл с настройками нашего приложения, который при старте работы получает следующие переменные окружения: индивидуальный токен бота, адрес электронной почты и пароль для почтового сервера.

```
# coding utf-8
import os
class Settings(object):
    """Настройки для бота."""
   token = '5177717001:AAGTumjR8O1C1TfbQXDpvzvO002O8tp_XL8'
   email = 'botmailpsu@gmail.com'
    mail_password = ('jmpqxiywcdevvvqq')
   def __str__(self):
       return 'Settings object'
```

Файл constants.py с константами для работы бота.

```
# coding: utf-8
"""Модуль для хранения констант."""
# flake8: noqa
START_HELP_MESSAGE = (
   'Доступные команды:\n'
   '/Help | /Start: информация об активных командах\n'
   '/Registration | /Регистрация: регистрация\n'
)
ADMIN_HELP_MESSAGE = (
   'База телефонов: просмотр входящих заявок в секторы студенческого совета\n'
   'Подача документов: вкладка для получения материальной помощи или активистской стипендии\n'
   'Создать новость: создание новостей для ленты бота.\n'
   'Загрузка шаблона для активистской стипендии: загрузка шаблона для подачи заявления на активистскую стипендию.\n'
   'Загрузка шаблона для получения материальной помощи: загрузка шаблона для подачи заявления на получение мат. помощи.\n'
```

```
        'Справка: вызов этого сообщения :).\n'
        'Новости: просмотр новостного блока.\n'
    )
    USER_HELP_MESSAGE = (
        'Подача документов: вкладка для получения материальной помощи или активистской стипендии\n'
        'Справка: информация о командах бота\n'
        'Вступить в студенческий совет: отправить заявку в студенческий совет для вступления в один из секторов\n'
        'Новости: узнать о последних новостях\n'
    )
    PHONE_SYMBOL_MAX = 12
    EMAIL_REGEX = (
     r'([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+'
    )
```

Файл enums.py с перечислениями для работы бота.

```
    class DocumentDestination(str, Enum):
        """Перечисление назначения документа."""
        grant = 'Получение активистской стипендии'
        material_aid = 'Получение материальной помощи'
        template_grant = 'Шаблон для активистской стипендии'
        template_material_aid = 'Шаблон для получения материальной помощи'
    class SectorName(str, Enum):
        """Названия секторов."""
        sport = 'Спортивный сектор'
        cultural = 'Культмассовый сектор'
        decorator = 'Сектор Декораторов'
        vocal = 'Вокальный сектор'
        volunteer = 'Волонтёрский сектор'
```

```
    design_creativity = 'Центр проектного творчества'
    head = 'Сектор старост'
    chairman_trade_union = 'Председатель Профсоюза'
    chairman_student_council = 'Председатель ССС'
```

Файл ai_handler.py, в который загружаются библиотеки для работы с перечислениями, с данными в формате CSV, для работы с чат-ботами на Python, а также библиотека для работы с API OpenAI. В данной работе используются библиотека enum, необходимая для определения перечислений, представляющих собой набор символических имен, библиотека time, используемая для создания задержек выполнения при ожидании выполнения операций. Библиотека aiogram необходима для асинхронной работы с ботами на базе telegram. Из библиотеки необходимо импортировать dispatcher – класс, необходимый для маршрутизации и обратки входящих сообщений и событий. Его задача – регистрация обработчиков для определенных типов сообщений. Также нужно импортировать types для работы с объектами и структурами данных из Telegram API. Для работы с ChatPGT 4 необходимо импортировать модуль Open AI из одноименной библиотеки. В рамках работы необходимо создать отдельный поток с созданным ассистентом, который будет опираться только на загруженный файл.

```
import enum
import time
from aiogram import Dispatcher, types
from openai import OpenAI
from config import Settings
```

После этого необходимо задать необходимые промпты в классе Prompts которые необходимы для изначальных настроек ИИ-помощнику. Промпт – обусловленный запрос к нейросети, запрошенный с целью получения ответа. В данной работе благодаря промптам производится первичная настройка интеллектуальной диалоговой системы.

```
class Prompts(enum.Enum):
```

```
        """Промты"""
        MSG_PROMPT = ("Ответь на это, учитывая эти правила:"
                      " Постарайся ответить раскрыто и без воды."
                      ""
                      "(res1mod1.csv)."
                      " Только если спросят твоё имя, говори, что
тебя зовут Bubba."
                      " Только если спросят где тебя сделали,"
                      " то тебя придумал Максим Митрофанов"
                      )
```

После этого необходимо создать класс MyOpenAI, состоящий из ряда методов, необходимых для интеграции ChatGPT 4 с интеллектуальной диалоговой системой вуза. Метод __init__ отвечает за инициализацию клиента OpenAI, устанавливая API-ключ доступа, а также путь к датасету, на основе которого чат-бот будет дополнительно выдавать ответы.

```
    class MyOpenAI:
        """Класс для взаимодействия с OpenAI"""
        def __init__(self):
            self.client = OpenAI(api_key=Settings.OPENAI_TOKEN)
            self.csv_file                                    =
"/home/williamcano/PycharmProjects/work/diploma/cwurData.csv"
```

Метод test_answer_gpt создан для получения ответа от ChatGPT. В системе указано ключевое слово, указывая которое система создает отдельный поток с сущностью ассистента, который опирается сугубо на загруженный файл. Если в теле письма, который отправляет пользователь данный ключ не используется – вызывается стандартный ИИ-помощник.

```
    async def text_answer_gpt(self, message: str):
            """Ответ текстом от ГПТ"""
            if "psu" in message.split(" "):
                res = await self.thread_run(
                    message,
                    thread="thread_E3m22YEwMNVH1lnq3p5A2tuN",
                    assistant_id="asst_qyUU2LBFQVv2q2aohxIpP3fs"
```

```
                )
                return res
            completion = self.client.chat.completions.create(
                model="gpt-4o",
                messages=[
                    {"role":         "system",         "content":
Prompts.MSG_PROMPT.value},
                    {"role": "user", "content": message}
                ],
            )
            return completion.choices[0].message.content
```

Метод upload_file необходим для конвертирования CSV-файла в двоичный код, чтобы в дальнейшем передать его на сервер OPENAI. На выходе мы получаем ID загруженного файла.

```
    def upload_file(self) -> str:
            """Загрузка файлов в OPenAI"""
            with open(self.csv_file, "rb") as binary_file:
                upload_file = self.client.files.create(
                    file=binary_file,
                    purpose="assistants"
                )
                return upload_file.id
```

После этого с помощью метода create_assistant необходимо создать ассистента, который на входе изначально получает конкретный датасет, на который будет опираться в виде идентификатора файла, загруженного ранее. В OPENAI можно обратиться как к главной модели, которая содержит в себе обширный датасет, так и создать отдельную версию и дополнительно обучить ее.

```
        def create_assistant(self, file_id: str):
            """Метод для создания ассистента"""
            assistant = self.client.beta.assistants.create(
                instructions=Prompts.MSG_PROMPT.value,
                model="gpt-4o",
```

```
                tools=[{"type": "code_interpreter"}],
                tool_resources={
                    "code_interpreter": {
                        "file_ids": [file_id]
                    }
                }
            )
            return assistant.id
```

Метод creatr_thread необходима для создания потока, необходимого для вызова ассистента. Метод возвращает идентификатор потока.

```
    def create_thread(self, file_id: str, message: str):
            """Метод для создания потока"""
            thread = self.client.beta.threads.create(
                messages=[
                    {
                        "role": "user",
                        "content": message,
                        "attachments": [
                            {
                                "file_id": file_id,
                                "tools":                [{"type":
"code_interpreter"}]
                            }
                        ]
                    }
                ]


            )
        return thread.id
```

Чтобы отправлять сообщения ассистенту в поток, необходимо реализовать метод thread_run, также в методе происходит ожидание ответа от ассистента после проверки статуса.

```
      async def thread_run(self, message: str, thread: str,
assistant_id: str):
```

```
            """Метод для отправки сообщения в поток с ассистентом
и файлом"""
            self.client.beta.threads.messages.create(
                thread,
                role="user",
                content=message
            )
            run = self.client.beta.threads.runs.create(
                thread_id=thread,
                assistant_id=assistant_id
            )
            get_status = self.client.beta.threads.runs.retrieve(
                run_id=run.id,
                thread_id=thread
            )
            while get_status.status != "completed" and run.status
!= "failed":
                time.sleep(5)
                get_status                                      =
self.client.beta.threads.runs.retrieve(
                    run_id=run.id,
                    thread_id=thread
                )
            assistant_response                                  =
self.client.beta.threads.messages.list(
                thread_id=thread
            )
            return
assistant_response.data[0].content[0].text.value
```

Благодаря методу ai_msg_response происходит проверка наличия текста в сообщении пользователя, после чего, если сообщение есть – получаем ответ от модели GPT на текст данного сообщения, после чего ответ отправляется пользователю.

```
    async def ai_msg_response(message: types.Message):
```

```
        """метод для обработки текста исходя из
         сообщения со вставкой промта и датасета"""
        if message.text:
            response = await
MyOpenAI().text_answer_gpt(message.text)
            await message.answer(response)
```

В методе ai_handlers происходит регистрация обработчиков диспетчера сообщений, который при регистрации сообщения вызывается функцию ai_msg_response

```
def ai_handlers(dispatcher: Dispatcher):
    """Регистрация всех хендлеров"""
    dispatcher.register_message_handler(
        callback=ai_msg_response
    )
```

Файл main.py, в который импортируется файлы admin.py и client.py, после чего вызываем функцию …register_handlers и передаем туда диспетчер бота. Это необходимо для регистрации всех обработчиков из вышеупомянутых файлов. С помощью функции executor.start_polling(dp, skip_updates=True, on_startup=on_startup) бот активируется, начинает работу и выходит в онлайн.

```
# coding utf-8
from aiogram.utils import executor
from handlers import admin, client
from start_bot import dp
async def on_startup(_):
   print('Бот вышел в онлайн.')
client.client_register_handlers(dispatcher=dp)
admin.admin_register_handlers(dispatcher=dp)
if __name__ == '__main__':
   executor.start_polling(dp, skip_updates=False,
on_startup=on_startup)
```

Файл models.py - файл с описанием таблиц базы данных. Необходим класс для описания моделей declarative_base() для работы с таблицами. Рассмотрим на примере таблицы User поля в базе данных: id (идентификатор), tg_user_id

(идентификатор пользователя в telegram), fio (ФИО), phone (телефон), email_adress (адрес электронной почты), status (), sector_name (название сектора). Кроме того, необходимо дать название таблице через __tablename__[8].

```
class User(Base):
    """Пользователь."""

    __tablename__ = 'user'

    id = Column(Integer, primary_key=True)
    tg_user_id = Column(Integer, nullable=False)
    fio = Column(String(60), nullable=False)
    phone = Column(String(PHONE_SYMBOL_MAX), nullable=False,
unique=True)
    email_address = Column(String(30), nullable=False,
unique=True)
    status = Column(String(40), nullable=False)
    sector_name = Column(String(40), nullable=True)
```

В файле start_bot.py проходит инициализация бота по токену, который находится в файле config.py, инициализация диспатчера, а также создание объекта движка и драйвера субд. Для того, чтобы сохранялись ответы, полученные от пользователя, необходимо использовать MemoryStorage для хранения в памяти сообщений между состояниями.

```
# coding: utf-8
from aiogram import Bot, Dispatcher
from aiogram.contrib.fsm_storage.memory import MemoryStorage
from sqlalchemy.ext.asyncio import AsyncSession,
create_async_engine
from config import Settings
memory = MemoryStorage()
config = Settings()
bot = Bot(token=config.token)
```

```
dp = Dispatcher(bot, storage=memory)
engine = create_async_engine('sqlite+aiosqlite:///psu_bot', future=True)
async_session = AsyncSession(engine, expire_on_commit=False)
```

Файл utilits.py с функциями в проекте. Одной из таких функций является функция is_admin(), благодаря который бот понимает, является ли пользователь администратором или нет.

```
async def is_admin(user_id: int, connection) -> bool:
    """Возвращает True, если пользователь администратор.
    :param user_id: уникальный идентификатор пользователя телеграм,
    :param connection: подключение к базе данных.
    """
    select_stmt = select(
        User.tg_user_id
    ).where(
        User.tg_user_id == user_id,
        User.status == RoleUsers.admin.value
    )
    admin_id = (await connection.execute(select_stmt)).first()
    return bool(admin_id)
```

Функции select и where, взятые из пакета sqlalchemy, аналогичны реальным SQL-запросам. Тем самым мы делаем запрос в нашу базу данных, где проверяем пользователя среди действующих администраторов.

Функция check_sector_name проверяет строку, вводимую пользователем при подаче заявки на вступление в активистский сектор. Все возможные названия зафиксированы, поэтому необходимо проверить, имеется ли введенный сектор в пуле секторов. В классе SectorName при помощи перечисления просматриваем поля класса (поле класса – сектор). Если введенная строка совпадает с названием сектора, то будет возвращено True, в противном случае – False.

```
async def check_sector_name(sector_name: str) -> bool:
```

```
    """Проверка ввода сектора."""
    return bool(sector_name in SectorName.get_choices())
```

Следующей функцией является phone_validator, необходимая для определения валидности номера телефона пользователя. Длина введенной строки сравнивается с переменной-константой PHONE_SYMBOL_MAX, если длина номера больше 12, то номер определяется как невалидный и сопровождается соответствующим сообщением. Также разработана проверка формата ввода номера телефона. Пользователь может указать номер с кода +7. В этом случае благодаря phone_number.strip('+') удаляем “+” и проверяем, является ли строка номером или набором случайных символов.

```
async def phone_validator(phone_number: str) -> tuple:
    """Валидация номера телефона."""
    is_valid = True
    help_message = ''
    if len(phone_number) > PHONE_SYMBOL_MAX:
        is_valid = False
         help_message = 'Неверная длина номера.\n'
    if not phone_number.strip('+').isnumeric():
        is_valid = False
         help_message += 'Обнаружены недопустимые символы.\n'
    return is_valid, help_message
```

Функция client_cancel является обработчиком кнопки отмены для пользователя. После того, как пользователь введет слово “Отмена”, либо нажмет на одноименную кнопку – ему вернется главное меню. Данная функция работает только с рядовыми пользователями, для работы с администраторами разработана функция admin_cancel.

```
async def client_cancel(message: types.message) -> bool:
    """Кнопка отмены для пользователя."""
    is_not_cancel = True
    keyboard = ReplyKeyboardRemove()
    if message.text == 'Отмена':
         stmt_user = select(
```

```
                User.id
            ).where(
                User.tg_user_id == message.from_user.id,
                User.status == RoleUsers.user.value,
            )
            user_id                    =                    (await
async_session.execute(stmt_user)).first()
            if user_id:
                keyboard = user_keyboard
            await bot.send_message(
                message.from_user.id,
               'Возвращаю в главное меню.',
               reply_markup=keyboard,
            )
             is_not_cancel = False
        return is_not_cancel

    async def admin_cancel(message: types.message) -> bool:
        """Кнопка отмены для админа."""
        is_not_cancel = True
        keyboard = ReplyKeyboardRemove()
        if message.text == 'Отмена':
            stmt_admin = select(
                User.tg_user_id
            ).where(
                User.tg_user_id == message.from_user.id,
                User.status == RoleUsers.admin.value
            )
            admin_id                    =                    (await
async_session.execute(stmt_admin)).first()
            if admin_id:
                keyboard = admin_keyboard
            await bot.send_message(
                message.from_user.id,
               'Возвращаю в главное меню.',
```

```
                reply_markup=keyboard,
            )
             is_not_cancel = False
        return is_not_cancel
```

В связи с тем, что у нашего бота есть две роли (пользователь и администратор), для каждой из них реализована своя клавиатура. Функция get_keyboard посылает запрос в базу данных с целью проверки ID мессенджера пользователя. В зависимости от того, какой ролью обладает студент, такая клавиатура и будем ему доступна.

```
    async def get_keyboard(user_id: int, connection) -> ReplyKeyboardMarkup:
        """Получение клавиатуры в общих методах."""
        keyboard = user_keyboard
        stmt_status = select(
            User.status
        ).where(
            User.tg_user_id == user_id,
        )
        status = (await connection.execute(stmt_status)).first()
        if status[0] == RoleUsers.admin.value:
            keyboard = admin_keyboard
        return keyboard
```

Функция, которая отвечает за передачу email-сообщения руководителю о подаче заявления студента на вступление в сектор. Благодаря smtplib.SMTP('smtp.gmail.com', 587 мы подключаемся к почтовому сервису gmail, а с помощью server.login(config.email, config.mail_password) – авторизуемся в почтовом сервисе путем взятия логина и пароля из файла config.py. После чего необходимо создать объект письма msg = MIMEMultupart(). Затем заполняем письмо и отправляем.

```
    async def send_mail(
            theme_mail: str,
            message_content: str,
```

```
            to_addr: str,
            files_paths=None,
    ) -> None:
        """Отправка письма на почту."""
        server = smtplib.SMTP('smtp.gmail.com', 587)
         server.starttls()
         server.login(config.email, config.mail_password)
        msg = MIMEMultipart()
        # Содержимое письма.
        if message_content:
             msg.attach(MIMEText(message_content, 'plain'))
        # Тема письма.
         msg['Subject'] = theme_mail
        if files_paths:
            for file_path in files_paths:
                with open(file_path, 'rb') as file:
                    # Отсекаем пути, получаем только имя файла.
                    head, filename = os.path.split(file_path)
                     part = MIMEApplication(
                         file.read(),
                         Name=filename,
                     )
                     # Закрепляем документ в письме.
                     part['Content-Disposition'] = (
                         f'attachment; filename={filename}'
                     )
                    msg.attach(part)
         server.sendmail(config.email,                    to_addr,
msg=msg.as_string())
        server.quit()
```

Файл admin.py, в котором хранятся обработчики сообщений и состояния приложения пользователя с ролью администратора. Например, функция start_admin_registration необходима для проверки роли администратора и для предложения зарегистрироваться. В ответ на команду /admin_reg срабатывает

этот хэндлер. После чего собирается запрос на проверку в таблице пользователя с telegram_id отправителя. Если такой пользователь уже есть – бот выдаст соответствующее сообщение. В ином случае начнется регистрация пользователя как администратора.

```
async def start_admin_registration(message: types.message):
    """Старт регистрации администраторов."""
    stmt = select(
        User
    ).where(
        User.tg_user_id == message.from_user.id,
    )
    user_exists = (await async_session.execute(stmt)).first()
    if not user_exists:
        await message.answer(
            'Напишите свою фамилию и имя через пробел:',
            reply_markup=cancel_keyboard,
        )
        await AdminRegistration.load_fio.set()
    else:
        keyboard = await get_keyboard(
            user_id=message.from_user.id,
            connection=async_session,
        )
        await message.answer(
            'Вы уже зарегистрированы.',
            reply_markup=keyboard,
        )
```

Файл client.py, в котором хранятся обработчики сообщений и состояния приложения пользователя с ролью администратора. Функционал практически идентичен функционалу, представленному в файле admin.py.

Файл ui.py, в котором реализованы все кнопки и клавиатура в целом. Кнопка создается с помощью KeyboardButton(SectorName.sport.value). Далее с помощью функции add() проходит сборка кнопок в клавиатуру. Важно

заполнить кнопки “лесенкой”, так как если внутри одной функции указать все кнопки – в клавиатуре они будут отображаться в линию.

```
    # Клавиатура выбора секторов.
    button_sport = KeyboardButton(SectorName.sport.value)
    button_cultural = KeyboardButton(SectorName.cultural.value)
    button_decorator = KeyboardButton(SectorName.decorator.value)
    button_vocal = KeyboardButton(SectorName.vocal.value)
    button_volunteer = KeyboardButton(SectorName.volunteer.value)
    button_design_creativity                                    =
KeyboardButton(SectorName.design_creativity.value)
    button_head = KeyboardButton(SectorName.head.value)

    sector_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
    sector_keyboard.add(
       button_sport,
    ).add(
       button_cultural,
    ).add(
       button_decorator,
    ).add(
       button_vocal,
    ).add(
       button_volunteer,
    ).add(
       button_design_creativity,
    ).add(
       button_head,
    ).add(
       button_cancel,
    )
```

ЗАКЛЮЧЕНИЕ

Результатом выполнения магистерской диссертации можно считать успешно реализованный в соответствии с целями интеллектуальную диалоговую систему вуза.

В результате был проведен глубокий анализ процессов коммуникаций и управления таким органом, как совет студенческого самоуправления. Были изучены проблемы руководителей активитстких подразделений, пожелания студентов-активистов. Была изучена проблема выбора университета абитуриентами. Перед началом разработки, также был проведен анализ всех возможных инструментов, а именно: ЯП, СУБД, библиотек для разработки диалоговых систем, мессенджеров для базирования бота, видов нейронных сетей, а также уже действующих ИИ-ассистентов.

По окончании работ было принято решение о разработке telegram-бота на языке программирования Python, с использованием в разработке системы управления базами данных PostgreSQL, с использованием ассинхронной библиотеки aiogram, предназначенной как раз таки для разработки ботов. Кроме того, средой разработки была выбрана Pycharm IDE.

На основе приведенных выше технологий, удалось спроектировать, разработать и протестировать приложение, со следующим функционалом:

– вызов помощника, готового отвечать на любые поставленные вопросы на основе ChatGPT,

– вызов помощника и его ответы на вопросы на основе заранее загруженного файла,

– отправка документов на получение материальных благ,

– предоставление возможности просмотра ленты новостей,

– отправление заявки на вступление в тот или иной сектор ССС,

– предоставление информационной помощи,

– загрузка новостных постов,

– загрузка шаблонов для последующей отправки ботов шаблона на получение материальных благ,

– просмотр базы телефонных номеров,

– оповещение руководителя сектора на электронную почту о входящей заявке.

В своей перспективе данная диалоговая система может послужить верным инструментом в рамках работы ССС. Планируется и в дальнейшем заниматься развитием проекта, добавить возможность автоматического заполнения ботом документов, возможность редактировать БД внутри бота, а также разработка собственной нейронной сети и собственного датасета по ряду тем, наиболее важных для студентов университетов.

Интеллектуальная диалоговая система вуза на основе telegram-бота готова к эксплуатации в новом учебном году.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Берд, Барри Java для чайников / Барри Берд. - М.: Диалектика / Вильямс, 2013 – С. 14-17.

2. Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с нгл. – СПб.: БХВ-Петербург, 2019 – С. 20-23.

3. Дэвис С. Р. C# 2005 для «чайников». – Издательство Диалектика, 2008 – С. 53-85.

4. Чурсин Г. С. Встраиваемый чат-бот в готовые корпоративные системы. – 2019 – С. 23-25.

5. Забутной А., Сокол М. Эволюция СУБД Oracle. – 2018 – С. 140-143.

6. Пономарев Д. А. Работа с базами данных при помощи языка программирования Python и СУБД SQLite3 в школьном курсе информатики //Информационные технологии в образовании. – 2020. – №. 3. – С. 187-191.

7. Тимошенко Л. И., Аракелов О. Г. Безопасность труда при работе с персональным компьютером //Культура и общество: история и современность. – 2014. – С. 114-117.

8. Карамышева Н. С., Митрофанов М. П., Акифьев И. В. Диалоговая система вуза на основе telegram-бота // Информатика: проблемы, методы, технологии: сборник материалов XXIII международной научно-практической конференции / под редакцией Д. Н. Борисова; Воронеж, Воронежский государственный университет, 15-17 февраля 2023 г. – Воронеж, ВГУ, 2023. – С. 1139-1147

ПРИЛОЖЕНИЕ А
(обязательное)

Интеллектуальная диалоговая система вуза

Листинг программы

ПРИЛОЖЕНИЕ А.1

Файл env.py

```
from logging.config import fileConfig

from sqlalchemy import engine_from_config
from sqlalchemy import pool

from alembic import context

# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
if config.config_file_name is not None:
    fileConfig(config.config_file_name)

# Meta config
from models import Base
target_metadata = Base.metadata

# other values from the config, defined by the needs of env.py,
# can be acquired:
#                     my_important_option                     =
config.get_main_option("my_important_option")
# ... etc.
```

```
    def run_migrations_offline():
        """Run migrations in 'offline' mode.

        This configures the context with just a URL
        and not an Engine, though an Engine is acceptable
        here as well.  By skipping the Engine creation
        we don't even need a DBAPI to be available.

        Calls to context.execute() here emit the given string to
the

        script output.

        """
        url = config.get_main_option("sqlalchemy.url")
        context.configure(
            url=url,
            target_metadata=target_metadata,
            literal_binds=True,
            dialect_opts={"paramstyle": "named"},
        )

        with context.begin_transaction():
            context.run_migrations()


    def run_migrations_online():
        """Run migrations in 'online' mode.

        In this scenario we need to create an Engine
        and associate a connection with the context.
```

```
        """
        connectable = engine_from_config(
            config.get_section(config.config_ini_section),
            prefix="sqlalchemy.",
            poolclass=pool.NullPool,
        )

        with connectable.connect() as connection:
            context.configure(
                connection=connection,
target_metadata=target_metadata
            )

            with context.begin_transaction():
                context.run_migrations()



    if context.is_offline_mode():
        run_migrations_offline()
    else:
        run_migrations_online()
```

ПРИЛОЖЕНИЕ А.2

Файл __init__.py

```
# coding: utf-8

from handlers import client

from handlers import admin
```

ПРИЛОЖЕНИЕ А.3

Файл admin.py

```
# coding: utf-8

import os

import re


from aiogram import Dispatcher, types

from aiogram.dispatcher import FSMContext

from aiogram.dispatcher.filters.state import State, StatesGroup

from aiogram.types import ReplyKeyboardRemove

from sqlalchemy import insert, select


from constans import EMAIL_REGEX

from enums import RoleUsers, DocumentDestination

from keyboards import cancel_keyboard

from keyboards.ui import (

    admin_keyboard,

    admin_sector_board,

    news_actions_keyboard, user_keyboard,

)

from models import News, User, Documents

from start_bot import async_session, bot
```

```
from utilits import (
    admin_cancel,
    is_admin,
    check_sector_name,
    phone_validator, convert_into_bytes, get_keyboard,
)



this_directory = os.getcwd()



class AdminRegistration(StatesGroup):
    """Состояния регистрации администраторов."""

    load_fio = State()
    load_sector = State()
    load_phone = State()
    load_email = State()



class CreateNew(StatesGroup):
    """Состояния создания новости."""
```

```
    add_header = State()
    add_content = State()



class GrantTemplateLoader(StatesGroup):
    """Состояние загрузки шаблона файла для активистской
стипендии."""

    load_file = State()



class MaterialAidTemplateLoader(StatesGroup):
    """Состояние загрузки файла."""

    load_file = State()



async def start_admin_registration(message: types.message):
    """Старт регистрации администраторов."""
    stmt = select(
        User
    ).where(
```

```
            User.tg_user_id == message.from_user.id,
        )
        user_exists = (await async_session.execute(stmt)).first()
        if not user_exists:
            await message.answer(
                'Напишите свою фамилию и имя через пробел:',
                reply_markup=cancel_keyboard,
            )
            await AdminRegistration.load_fio.set()
        else:
            keyboard = await get_keyboard(
                user_id=message.from_user.id,
                connection=async_session,
            )
            await message.answer(
                'Вы уже зарегистрированы.',
                reply_markup=keyboard,
            )


async def load_fio(message: types.message, state: FSMContext):
    """Загрузка фамилии и имени администратора."""
```

```
    if await admin_cancel(message=message):
        async with state.proxy() as data:
            data['fio'] = message.text
            await AdminRegistration.next()
            await message.answer(
                'Напишите название сектора, который Вы
курируете:',
                reply_markup=admin_sector_board,
            )
    else:
        await state.finish()



async def load_sector(message: types.message, state: FSMContext):
    """Загрузка сектора."""
    if await admin_cancel(message=message):
        if await check_sector_name(sector_name=message.text):
            check_stmt = select(
                User.sector_name
            ).where(
                User.sector_name == message.text
            )
            sector_name_exists = (
```

```
                await async_session.execute(check_stmt)
            ).first()
            if not sector_name_exists:
                async with state.proxy() as data:
                    data['sector_name'] = message.text
                    await AdminRegistration.next()
                    await message.answer(
                        'Напишите свой телефонный номер:',
                        reply_markup=cancel_keyboard,
                    )
            else:
                await message.answer(
                    'Данный сектор уже курирует другой
пользователь',
                    reply_markup=admin_sector_board,
                )
                await AdminRegistration.load_sector.set()
        else:
            await AdminRegistration.load_sector.set()
            await message.answer(
                f'Непримелимое название сектора
{message.text}.\n'
```

```
                f'Напишите название сектора, который Вы
курируете:',
                reply_markup=admin_sector_board,
            )
    else:
        await state.finish()



async def load_phone(message: types.message, state: FSMContext):
    """Загрузка телефонного номера."""
    if await admin_cancel(message=message):
        is_valid, help_message = await phone_validator(
            phone_number=message.text
        )
        if not is_valid:
            await AdminRegistration.load_phone.set()
            await message.reply(f'{help_message}Повторите
попытку:')
        else:
            stmt_phone = select(
                User.phone
            ).where(
                User.phone == message.text,
```

```
            )
            phone_exists = (await
async_session.execute(stmt_phone)).first()
            if not phone_exists:
                async with state.proxy() as data:
                    data['phone'] = message.text
                await message.answer(
                    'Укажите свою электронную почту:',
                    reply_markup=cancel_keyboard,
                )
                await AdminRegistration.next()
            else:
                await message.answer(
                    'Этот номер уже используется! Укажите
другой.',
                    reply_markup=cancel_keyboard,
                )
                await AdminRegistration.load_phone.set()
    else:
        await state.finish()



async def load_email(message: types.message, state: FSMContext):
```

```
"""Загрузка электронной почты."""
if await admin_cancel(message=message):
    regex = re.compile(EMAIL_REGEX)
    if re.fullmatch(regex, message.text):
        async with state.proxy() as data:
            data['email_address'] = message.text
            email_stmt = select(
                User.email_address
            ).where(
                User.email_address == message.text
            )
            email_exists = (
                await async_session.execute(email_stmt)
            ).first()
            if not email_exists:
                insert_stmt = insert(
                    User
                ).values(
                    fio=data['fio'],
                    sector_name=data['sector_name'],
                    email_address=data['email_address'],
                    tg_user_id=message.from_user.id,
```

```
                        status=RoleUsers.admin.value,
                        phone=data['phone'],
                    )
                    await async_session.execute(insert_stmt)
                    await async_session.commit()
                    await state.finish()
                    await message.answer(
                        'Регистрация успешно завершена.',
                        reply_markup=admin_keyboard,
                    )
                else:
                    await message.reply(
                        'Указанная почта уже занята, укажите
другую:',
                        reply_markup=cancel_keyboard,
                    )
                    await AdminRegistration.load_email.set()
        else:
            await AdminRegistration.load_email.set()
            await message.reply(
                'Неверный формат. Введите электронную почту:'
            )
    else:
```

```
        await state.finish()



async def phone_numbers(message: types.message):
    """Вывод телефонных номеров."""
    if await is_admin(
            user_id=message.from_user.id,
            connection=async_session,
    ):
        answer_message = ''
        all_stmt = select(
            User.fio,
            User.phone
        ).where(
            User.status == RoleUsers.user.value,
        )
        students = (await async_session.execute(all_stmt)).all()
        if students:
            for fio, phone in students:
                answer_message += f'{fio} {phone}\n'
        else:
            answer_message = 'Пока нет ни одной заявки.'
```

```
        await message.answer(answer_message)



async def create_new(message: types.message, state: FSMContext):
    """Переход в состояние создания новости."""
    if await is_admin(
            user_id=message.from_user.id,
            connection=async_session
    ):
        if await admin_cancel(message=message):
            await message.answer(
                'Напишите заголовок новости:',
                reply_markup=cancel_keyboard,
            )
            await CreateNew.next()



async def create_header_news(message: types.message, state:
FSMContext):
    """Создание заголовка новости."""
    if await is_admin(
            user_id=message.from_user.id,
            connection=async_session
```

```
    ):
        if await admin_cancel(message=message):
            async with state.proxy() as data:
                data['header'] = message.text
                await CreateNew.next()
                await message.answer(
                    'Напишите, о чём Вы хотите рассказать?',
                    reply_markup=cancel_keyboard,
                )
        else:
            await state.finish()



async def create_content_news(message: types.message, state:
FSMContext):
    """Создание содержания новости."""
    if await is_admin(
            user_id=message.from_user.id,
            connection=async_session
    ):
        if await admin_cancel(message=message):
            async with state.proxy() as data:
                data['content'] = message.text
```

```
                insert_stmt = insert(
                    News
                ).values(
                    header=data['header'],
                    content=data['content'],
                    creator=message.from_user.id,
                )
                await async_session.execute(insert_stmt)
                await async_session.commit()
                await message.answer(
                    'Новость успешно создана!',
                    reply_markup=admin_keyboard,
                )
        await state.finish()



async def load_grant_templates_preview(message: types.message):
    """Подменю загрузки шаблона документа активисткой стипендии."""
    if await is_admin(user_id=message.from_user.id, connection=async_session):
        if await admin_cancel(message=message):
            await message.answer(
```

```
                'Жду шаблон в формате pdf.',
                reply_markup=ReplyKeyboardRemove(),
            )
            await GrantTemplateLoader.load_file.set()


async def load_grant_templates(message: types.document, state:
FSMContext):
    """Загрузка шаблона документа активисткой стипендии."""
    if re.search(r'\d*.pdf', message.document.file_name):
        downloaded_file = await bot.download_file_by_id(
            message.document.file_id,
            destination_dir=this_directory,
        )
        new_file_path = downloaded_file.name
        # Отсекаем абсолютный путь, получаем только имя файла.
        file_name = re.search(r'file_\d*.pdf', new_file_path)
        # Aiogram загружает файлы на сервер под именами
        # file_<порядковый номер>.<формат>, потому нужна именно
такая маска.
        if file_name:
            file_name = file_name.group()
            byte_code = await convert_into_bytes(
```

```
                f'{this_directory}/documents/{file_name}'
            )
            insert_docs = insert(
                Documents
            ).values(
                document=byte_code,
                file_name=message.document.file_name,
                owner=message.from_user.id,

destination=DocumentDestination.template_grant.value,
            )
            await async_session.execute(insert_docs)
            await async_session.commit()
            # Теперь удалим файл, который мы записали в бд с
сервера.
            os.remove(f'{this_directory}/documents/{file_name}')
            await message.answer(
                f'Файл {message.document.file_name} успешно
загружен',
                reply_markup=admin_keyboard,
            )
            await state.finish()
```

```
async def load_material_aid_templates_preview(message:
types.message):
    """Подменю загрузки шаблона документа для получения
материальной помощи."""
    if await is_admin(user_id=message.from_user.id,
connection=async_session):
        if await admin_cancel(message=message):
            await message.answer(
                'Жду шаблон в формате pdf.',
                reply_markup=ReplyKeyboardRemove(),
            )
            await MaterialAidTemplateLoader.load_file.set()



async def load_material_aid_templates(
        message: types.document,
        state: FSMContext,
):
    """Загрузка шаблона документа для получения материальной
помощи."""
    if re.search(r'\d*.pdf', message.document.file_name):
        downloaded_file = await bot.download_file_by_id(
            message.document.file_id,
```

```
            destination_dir=this_directory,
        )
        new_file_path = downloaded_file.name
        # Отсекаем абсолютный путь, получаем только имя файла.
        file_name = re.search(r'file_\d*.pdf', new_file_path)
        # Aiogram загружает файлы на сервер под именами
        # file_<порядковый номер>.<формат>, потому нужна именно
такая маска.
        if file_name:
            file_name = file_name.group()
            byte_code = await convert_into_bytes(
                f'{this_directory}/documents/{file_name}'
            )
            insert_docs = insert(
                Documents
            ).values(
                document=byte_code,
                file_name=message.document.file_name,
                owner=message.from_user.id,

destination=DocumentDestination.template_material_aid.value,
            )
            await async_session.execute(insert_docs)
```

```
            await async_session.commit()
            # Теперь удалим файл, который мы записали в бд с
сервера.
            os.remove(f'{this_directory}/documents/{file_name}')
            await message.answer(
                f'Файл {message.document.file_name} успешно
загружен',
                reply_markup=admin_keyboard,
            )
            await state.finish()



def admin_register_handlers(dispatcher: Dispatcher):
    """Регистрация всех хэндлеров."""
    dispatcher.register_message_handler(
        start_admin_registration,
        commands=['Admin_reg'],
        state=None,
    )
    dispatcher.register_message_handler(
        load_fio,
        state=AdminRegistration.load_fio,
    )
```

```
dispatcher.register_message_handler(
    load_sector,
    state=AdminRegistration.load_sector,
)
dispatcher.register_message_handler(
    load_email,
    state=AdminRegistration.load_email,
)
dispatcher.register_message_handler(
    load_phone,
    state=AdminRegistration.load_phone,
)
dispatcher.register_message_handler(
    phone_numbers,
    regexp='База телефонов',
)
dispatcher.register_message_handler(
    create_new,
    regexp='Создать новость',
    state=None,
)
dispatcher.register_message_handler(
```

```
        create_header_news,
        state=CreateNew.add_header,
    )
    dispatcher.register_message_handler(
        create_content_news,
        state=CreateNew.add_content,
    )
    dispatcher.register_message_handler(
        load_grant_templates_preview,
        regexp='Загрузка шаблона для активистской стипендии',
    )
    dispatcher.register_message_handler(
        load_grant_templates,
        state=GrantTemplateLoader.load_file,
        content_types=[types.ContentType.DOCUMENT],
    )
    dispatcher.register_message_handler(
        load_material_aid_templates_preview,
        regexp='Загрузка шаблона для получения материальной
помощи',
    )
    dispatcher.register_message_handler(
        load_material_aid_templates,
```

```
        state=MaterialAidTemplateLoader.load_file,
        content_types=[types.ContentType.DOCUMENT],
    )
```

ПРИЛОЖЕНИЕ А.4

Файл client.py

```
# coding: utf-8
import os
import re


from aiogram import Dispatcher, types
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters.state import State, StatesGroup
from aiogram.types import ReplyKeyboardRemove
from sqlalchemy import insert, select


from constans import ADMIN_HELP_MESSAGE, EMAIL_REGEX,
START_HELP_MESSAGE, \
    USER_HELP_MESSAGE
from enums import RoleUsers, DocumentDestination, SectorName
from keyboards.ui import (
    admin_keyboard,
    cancel_keyboard,
```

```
    help_keyboard,
    sector_keyboard,
    user_keyboard,
)
from models import News, StudentCouncilApplication, User,
Documents
from start_bot import async_session, bot
from utilits import (
    is_admin,
    check_sector_name,
    client_cancel,
    phone_validator,
    send_mail, convert_into_bytes, get_keyboard, write_to_file,
)



this_directory = os.getcwd()



class RegistrationStates(StatesGroup):
    """Состояния регистрации."""

    load_fio = State()
```

```
    load_phone = State()
    load_email = State()


class StatestJoinStudentCouncil(StatesGroup):
    """Состояния вступления в студенческий совет."""

    sector_name = State()
    skill = State()
    new_skill = State()


class UserStates(StatesGroup):
    """Состояния пользователя."""

    submission_of_documents = State()
    faq = State()
    news = State()
    join_student_council = State()


class ActivistHelp(StatesGroup):
```

```
    preview = State()
    load_files_for_material_aid = State()
    load_files_for_activist_help = State()



async def start_registration(message: types.message):
    """Старт регистрации пользователей."""
    select_stmt = select(
        User.tg_user_id
    ).where(
        User.tg_user_id == message.from_user.id
    )
    user_exists = (await
async_session.execute(select_stmt)).first()
    if not user_exists:
        await message.answer(
            'Напишите свою фамилию и имя через пробел:',
            reply_markup=cancel_keyboard,
        )
        await RegistrationStates.load_fio.set()
    else:
        await message.answer(
```

```
            'Вы уже зарегистрированы.',
            reply_markup=user_keyboard,
        )


async def load_fio(message: types.message, state: FSMContext):
    """Загрузка фамилии и имени."""
    if await client_cancel(message=message):
        async with state.proxy() as data:
            data['fio'] = message.text
            await RegistrationStates.next()
            await message.answer(
                'Напишите свой телефонный номер:',
                reply_markup=cancel_keyboard,
            )
    else:
        await state.finish()


async def load_phone(message: types.message, state: FSMContext):
    """Загрузка телефонного номера."""
    if await client_cancel(message=message):
```

```
        is_valid, help_message = await phone_validator(
            phone_number=message.text
        )
        if not is_valid:
            await RegistrationStates.load_phone.set()
            await message.reply(f'{help_message}Повторите
попытку:')
        else:
            stmt_phone = select(
                User.phone
            ).where(
                User.phone == message.text,
            )
            phone_exists = (await
async_session.execute(stmt_phone)).first()
            if not phone_exists:
                async with state.proxy() as data:
                    data['phone'] = message.text
                await message.answer(
                    'Укажите свою электронную почту:',
                    reply_markup=cancel_keyboard,
                )
                await RegistrationStates.next()
```

```
            else:
                await message.answer(
                    'Этот номер уже используется! Укажите
другой.',
                    reply_markup=cancel_keyboard,
                )
                await RegistrationStates.load_phone.set()
    else:
        await state.finish()



async def load_user_email(message: types.message, state:
FSMContext):
    """Загрузка электронной почты."""
    if await client_cancel(message=message):
        regex = re.compile(EMAIL_REGEX)
        if re.fullmatch(regex, message.text):
            async with state.proxy() as data:
                data['email'] = message.text
                email_stmt = select(
                    User.email_address
                ).where(
                    User.email_address == message.text
```

```
)
email_exists = (
    await async_session.execute(email_stmt)
).first()
if not email_exists:
    insert_stmt = insert(
        User
    ).values(
        fio=data['fio'],
        phone=data['phone'],
        email_address=data['email'],
        tg_user_id=message.from_user.id,
        status=RoleUsers.user.value,
        sector_name=None,
    )
    await async_session.execute(insert_stmt)
    await async_session.commit()
    await state.finish()
    await message.answer(
        'Регистрация успешно завершена.',
        reply_markup=user_keyboard,
    )
```

```
                else:
                    await message.reply(
                        'Указанная почта уже занята, напишите
другую:',
                        reply_markup=cancel_keyboard,
                    )
                    await RegistrationStates.load_email.set()
        else:
            await RegistrationStates.load_email.set()
            await message.answer(
                'Неверный формат. Введите электронную почту:',
                reply_markup=cancel_keyboard,
            )
    else:
        await state.finish()



async def start_message(message: types.message):
    """Меню вспомогательных сообщений."""
    select_stmt = select(
        User.status
    ).where(
        User.tg_user_id == message.from_user.id,
```

```
    )
    status = (await async_session.execute(select_stmt)).first()
    if status:
        if status[0] == RoleUsers.admin.value:
            await message.answer(
                ADMIN_HELP_MESSAGE,
                reply_markup=admin_keyboard,
            )
        else:
            await message.answer(
                USER_HELP_MESSAGE,
                reply_markup=user_keyboard,
            )
    else:
        await message.answer(
            START_HELP_MESSAGE,
            reply_markup=ReplyKeyboardRemove(),
        )



async def start_join_student_council(message: types.message):
    """Начало вступления в студенческий совет."""
```

```
    if await is_admin(
        user_id=message.from_user.id,
        connection=async_session,
    ):
        await message.answer(
            'Администратор не может подавать заявку на вступление
в студ.совет',
            reply_markup=admin_keyboard,
        )
    else:
        await StatestJoinStudentCouncil.sector_name.set()
        await message.answer(
            'В какой сектор желаете попасть?',
            reply_markup=sector_keyboard,
        )



async def load_sector_for_council(message: types.message, state:
FSMContext):
    """Загрузка сектора для студенческого совета."""
    if await client_cancel(message=message):
        if await check_sector_name(sector_name=message.text):
            async with state.proxy() as data:
```

```
                data['sector_name'] = message.text
                await StatestJoinStudentCouncil.next()
                await message.answer(
                    'Напишите, что Вы умеете делать?',
                    reply_markup=cancel_keyboard,
                )
        else:
            await message.answer(
                'Неверное название сектора. Попробуйте ещё
раз:\n',
                reply_markup=sector_keyboard,
            )
    else:
        await state.finish()



async def load_skill(message: types.message, state: FSMContext):
    """Загрузка текущих навыков для студенческого совета."""
    if await client_cancel(message=message):
        async with state.proxy() as data:
            data['skill'] = message.text
            await StatestJoinStudentCouncil.next()
```

```
            await message.answer('Напишите, чему Вы хотите
научиться?')
    else:
        await state.finish()



async def load_new_skill(message: types.message, state:
FSMContext):
    """Загрузка новых навыков для студенческого совета."""
    if await client_cancel(message=message):
        select_stmt = select(
            User.fio,
            User.phone,
            User.email_address
        ).where(
            User.tg_user_id == message.from_user.id,
        )
        user = (await async_session.execute(select_stmt)).first()
        if not user:
            await message.answer(
                'Кажется, Вы забыли зарегистрироваться.',
                reply_markup=user_keyboard,
            )
```

```
else:
    async with state.proxy() as data:
        data['new_skill'] = message.text
        sector_name = data['sector_name']
        insert_stmt = insert(
            StudentCouncilApplication
        ).values(
            sector_name=data['sector_name'],
            skill=data['skill'],
            new_skill=data['new_skill'],
            tg_user_id=message.from_user.id,
        )
        await async_session.execute(insert_stmt)
        await async_session.commit()
        await message.answer(
            'Заявка успешно отправлена.',
            reply_markup=user_keyboard,
        )
        admin_sector_stmt = select(
            User.email_address
        ).where(
            User.sector_name == sector_name
```

```
                )
                admin_email = (
                    await
async_session.execute(admin_sector_stmt)
                ).first()
                # Если руководитель сектора уже зарегистрирован,
                # отправим ему письмо на почту.
                if admin_email:
                    mail_body = (
                        f'Студент {user.fio} подал заявку на
вступление в '
                        f'{sector_name}. Вот такие навыки он
указал:\n'
                        f'{data["skill"]}. Желает научиться:\n'
                        f'{data["new_skill"]}. Номер телефона:
{user.phone}'
                    )
                    mail_header = (
                        f'Новая заявка на вступление в сектор
{sector_name}'
                    )
                    await send_mail(
                        theme_mail=mail_header,
                        message_content=mail_body,
```

```
                        to_addr=admin_email[0],
                    )
    await state.finish()



async def get_news(message: types.message):
    """Получить новостную ленту."""
    stmt_user = select(
        User.status
    ).where(
        User.tg_user_id == message.from_user.id
    )
    user_status = (await
async_session.execute(stmt_user)).first()
    if user_status:
        stmt_news = select(
            News.header,
            News.content,
        )
        news = (await async_session.execute(stmt_news)).all()
        keyboard = user_keyboard
        if user_status[0] == RoleUsers.admin.value:
            keyboard = admin_keyboard
```

```
        if news:
            for header, content in news:
                await message.answer(
                    f'{header}\n{content}',
                    reply_markup=keyboard,
                )
        else:
            await message.answer(
                'Пока нет новостей',
                reply_markup=keyboard,
            )


async def start_load_files_states(message: types.message):
    """Старт состояния подачи заявления на материальную помощь."""
    user_stmt = select(
        User.id
    ).where(
        User.tg_user_id == message.from_user.id,
    )
    user_exists = (await async_session.execute(user_stmt)).first()
```

```
    if user_exists:
        await ActivistHelp.preview.set()
        await message.answer(
            'Выберите, какую поддержку Вы хотите получить?:\n'
            'Здесь буду шаблоны документов',
            reply_markup=help_keyboard,
        )



async def help_menu_for_activist(
        message: types.message,
        state: FSMContext,
):
    """Состояние подменю для загрузки файлов."""
    if message.text != 'Отмена':
        if message.text == 'Материальная помощь':
            await message.answer(
                'Здесь будет информация о получении материальной
помощи\nШаблоны:',
                reply_markup=ReplyKeyboardRemove(),
            )
            stmt_docs = select(
                Documents
```

```
            ).where(
                Documents.destination == (
DocumentDestination.template_material_aid.value
                )
            )
            doc = (await
async_session.execute(stmt_docs)).first()
            if doc:
                # Перекодируем байты из бд в файл:
                await write_to_file(
                    binary_data=doc._data[0].document,
                    file_name=doc._data[0].file_name,
                )
                await message.reply_document(
                    document=open(
f'{this_directory}/{doc._data[0].file_name}', 'rb'
                    )
                )
os.remove(f'{this_directory}/{doc._data[0].file_name}')
            await ActivistHelp.next()
        elif message.text == 'Активистская стипендия':
```

```
            await message.answer((
                'Здесь будет информация о получении активистской
стипендии\n'
                'Шаблоны:'
                ),
            )
            stmt_docs = select(
                Documents
            ).where(
                Documents.destination == (
                    DocumentDestination.template_grant.value
                )
            )
            doc = (await
async_session.execute(stmt_docs)).first()
            if doc:
                # Перекодируем байты из бд в файл:
                await write_to_file(
                    binary_data=doc._data[0].document,
                    file_name=doc._data[0].file_name,
                )
                await message.reply_document(
                    document=open(
```

```
f'{this_directory}/{doc._data[0].file_name}', 'rb'
                    )
                )

os.remove(f'{this_directory}/{doc._data[0].file_name}')
            await ActivistHelp.load_files_for_activist_help.set()
        else:
            await message.answer(
                'Выберите один из вариантов:',
                reply_markup=help_keyboard,
            )
            await ActivistHelp.preview.set()
    else:
        stmt = select(
            User.status
        ).where(
            User.tg_user_id == message.from_user.id,
        )
        status = (await async_session.execute(stmt)).first()
        keyboard = user_keyboard
        if status:
            if status[0] == RoleUsers.admin.value:
```

```
            keyboard = admin_keyboard
        await message.answer(
            'Возвращаю в главное меню',
            reply_markup=keyboard
        )
        await state.finish()



async def load_files_for_activist_help(
        message: types.document,
        state: FSMContext
):
    """Загрузка файлов для получения активистской стипендии."""
    # Выбираем только файлы нужного нам формата:
    if re.search(r'\d*.pdf', message.document.file_name):
        downloaded_file = await bot.download_file_by_id(
            message.document.file_id,
            destination_dir=this_directory,
        )
        new_file_path = downloaded_file.name
        # Отсекаем абсолютный путь, получаем только имя файла.
        file_name = re.search(r'file_\d*.pdf', new_file_path)
```

```
        if file_name:
            file_name = file_name.group()
            byte_code = await convert_into_bytes(
                f'{this_directory}/documents/{file_name}'
            )
            insert_docs = insert(
                Documents
            ).values(
                document=byte_code,
                file_name=message.document.file_name,
                owner=message.from_user.id,
                destination=DocumentDestination.grant.value,
            )
            await async_session.execute(insert_docs)
            await async_session.commit()
            stmt_email = select(
                User.email_address
            ).where(
                User.sector_name ==
SectorName.chairman_trade_union.value,
            )
            email_address = (await
async_session.execute(stmt_email)).first()
```

```
            if email_address:
                await send_mail(
                    theme_mail='Заявка на получение активисткой
стипендии',
                    message_content='',
                    to_addr=email_address[0],

files_paths=[f'{this_directory}/documents/{file_name}', ],
                )
            # Теперь удалим файл, который мы записали в бд с
сервера.
            os.remove(f'{this_directory}/documents/{file_name}')
            keyboard = await get_keyboard(
                user_id=message.from_user.id,
                connection=async_session,
            )
            await message.answer(
                f'Файл {message.document.file_name} загружен',
                reply_markup=keyboard,
            )
            await state.finish()
```

```
async def load_files_for_material_aid_help(
        message: types.document,
        state: FSMContext
):
    """Загрузка файлов для получения материальной помощи."""
    # Выбираем только файлы нужного нам формата:
    if re.search(r'\d*.pdf', message.document.file_name):
        downloaded_file = await bot.download_file_by_id(
            message.document.file_id,
            destination_dir=this_directory,
        )
        new_file_path = downloaded_file.name
        # Отсекаем абсолютный путь, получаем только имя файла.
        file_name = re.search(r'file_\d*.pdf', new_file_path)
        if file_name:
            file_name = file_name.group()
            byte_code = await convert_into_bytes(
                f'{this_directory}/documents/{file_name}'
            )
            insert_docs = insert(
                Documents
            ).values(
```

```
                document=byte_code,
                file_name=message.document.file_name,
                owner=message.from_user.id,

destination=DocumentDestination.material_aid.value,
            )
            await async_session.execute(insert_docs)
            await async_session.commit()
            stmt_email = select(
                User.email_address
            ).where(
                User.sector_name ==
SectorName.chairman_trade_union.value,
            )
            email_address = (await
async_session.execute(stmt_email)).first()
            if email_address:
                await send_mail(
                    theme_mail='Заявка на получение помощи',
                    message_content='',
                    to_addr=email_address[0],

files_paths=[f'{this_directory}/documents/{file_name}', ],
                )
```

```
            # Теперь удалим файл, который мы записали в бд с
сервера.
            os.remove(f'{this_directory}/documents/{file_name}')
            keyboard = await get_keyboard(
                user_id=message.from_user.id,
                connection=async_session,
            )
            await message.answer(
                f'Файл {message.document.file_name} загружен',
                reply_markup=keyboard,
            )
            await state.finish()



async def get_faq(message: types.message):
    """Обработчик команды Справка."""
    select_stmt = select(
        User.status
    ).where(
        User.tg_user_id == message.from_user.id,
    )
    status = (await async_session.execute(select_stmt)).first()
    if status:
```

```
        if status[0] == RoleUsers.admin.value:
            await message.answer(
                ADMIN_HELP_MESSAGE,
                reply_markup=admin_keyboard,
            )
        else:
            await message.answer(
                USER_HELP_MESSAGE,
                reply_markup=user_keyboard,
            )
    else:
        await message.answer(
            START_HELP_MESSAGE,
        )



def client_register_handlers(dispatcher: Dispatcher):
    """Регистрация всех хэндлеров."""
    dispatcher.register_message_handler(
        start_message,
        commands=['Help', 'Start', 'Помощь'],
    )
```

```
dispatcher.register_message_handler(
    start_registration,
    commands=['Регистрация', 'Registration'],
    state=None,
)
dispatcher.register_message_handler(
    load_fio,
    state=RegistrationStates.load_fio,
)
dispatcher.register_message_handler(
    load_phone,
    state=RegistrationStates.load_phone,
)
dispatcher.register_message_handler(
    load_user_email,
    state=RegistrationStates.load_email,
)
dispatcher.register_message_handler(
    start_join_student_council,
    regexp='Вступить в студенческий совет',
    state=None,
)
```

```
    dispatcher.register_message_handler(
        load_sector_for_council,
state=StatestJoinStudentCouncil.sector_name,
    )
    dispatcher.register_message_handler(
        load_skill, state=StatestJoinStudentCouncil.skill,
    )
    dispatcher.register_message_handler(
        load_new_skill,
        state=StatestJoinStudentCouncil.new_skill,
    )
    dispatcher.register_message_handler(
        start_load_files_states,
        regexp='Подача документов',
        state=None,
    )
    dispatcher.register_message_handler(
        help_menu_for_activist,
        state=ActivistHelp.preview,
    )
    dispatcher.register_message_handler(
        load_files_for_activist_help,
        state=ActivistHelp.load_files_for_activist_help,
```

```
        content_types=[types.ContentType.DOCUMENT],
    )
    dispatcher.register_message_handler(
        load_files_for_material_aid_help,
        state=ActivistHelp.load_files_for_material_aid,
        content_types=[types.ContentType.DOCUMENT],
    )
    dispatcher.register_message_handler(
        get_news,
        regexp='Новости',
    ),
    dispatcher.register_message_handler(
        get_faq,
        regexp='Справка',
    )
```

ПРИЛОЖЕНИЕ А.5

Файл ui.py

```
# coding: utf-8
from aiogram.types import KeyboardButton, ReplyKeyboardMarkup


from enums import SectorName
```

```
# Клавиатура завершения состояний.
button_cancel = KeyboardButton('Отмена')


cancel_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
cancel_keyboard.add(button_cancel)


# Клавиатура выбора секторов.
button_sport = KeyboardButton(SectorName.sport.value)
button_cultural = KeyboardButton(SectorName.cultural.value)
button_decorator = KeyboardButton(SectorName.decorator.value)
button_vocal = KeyboardButton(SectorName.vocal.value)
button_volunteer = KeyboardButton(SectorName.volunteer.value)
button_design_creativity =
KeyboardButton(SectorName.design_creativity.value)
button_head = KeyboardButton(SectorName.head.value)


sector_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
sector_keyboard.add(
    button_sport,
).add(
    button_cultural,
).add(
    button_decorator,
```

```
).add(
    button_vocal,
).add(
    button_volunteer,
).add(
    button_design_creativity,
).add(
    button_head,
).add(
    button_cancel,
)


# Клавиатура секторов для администратора
button_trade_union =
KeyboardButton(SectorName.chairman_trade_union.value)
button_chairman =
KeyboardButton(SectorName.chairman_student_council.value)


admin_sector_board = ReplyKeyboardMarkup(resize_keyboard=True)
admin_sector_board.add(
    button_sport,
).add(
    button_cultural,
```

```
).add(
    button_decorator,
).add(
    button_vocal,
).add(
    button_volunteer,
).add(
    button_design_creativity,
).add(
    button_head,
).add(
    button_trade_union
).add(
    button_chairman
).add(
    button_cancel,
)


# Клавиатура пользователя.
button_documents = KeyboardButton('Подача документов')
button_news = KeyboardButton('Новости')
button_student_council = KeyboardButton('Вступить в студенческий
совет')
```

```
button_faq = KeyboardButton('Справка')


user_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
user_keyboard.row(
    button_documents,
    button_faq,
).add(
    button_student_council,
).row(
    button_news,
)


# Клавиатура состояний новостей.
button_create_new = KeyboardButton('Создать новость')


news_actions_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
news_actions_keyboard.add(
    button_create_new
).add(
    button_cancel
)
```

```
# Клавиатура для подачи документов.
button_help = KeyboardButton('Материальная помощь')
button_grants = KeyboardButton('Активистская стипендия')


help_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
help_keyboard.add(
    button_help
).add(
    button_grants
).add(
    button_cancel
)


# Клавиатура администратора.
button_news_admin = KeyboardButton('Создать новость')
button_phones = KeyboardButton('База телефонов')
button_admin_reg = KeyboardButton('Admin_reg')
button_load_template_grant = KeyboardButton(
    'Загрузка шаблона для активистской стипендии'
)
button_load_material_aid = KeyboardButton(
    'Загрузка шаблона для получения материальной помощи'
```

```
)

button_sql_runner = KeyboardButton('SQL-конструктор')


admin_keyboard = ReplyKeyboardMarkup(resize_keyboard=True)

admin_keyboard.add(

    button_phones, button_news_admin

).add(

    button_faq, button_news

).add(

    button_documents

).add(

    button_load_template_grant

).add(

    button_load_material_aid

)
```

ПРИЛОЖЕНИЕ А.6

Файл config.py

```
# coding utf-8

import os

class Settings(object):

    """Настройки для бота."""

    token = '5177717001:AAGTumjR8O1C1TfbQXDpvzvO002O8tp_XL8'
```

```
    email = 'botmailpsu@gmail.com'
    mail_password = ('jmpqxiywcdevvvqq')


    def __str__(self):
        return 'Settings object'
```

ПРИЛОЖЕНИЕ А.7

Файл constans.py

```
# coding: utf-8
"""Модуль для хранения констант."""
# flake8: noqa
START_HELP_MESSAGE = (
    'Доступные команды:\n'
    '/Help | /Start: информация об активных командах\n'
    '/Registration | /Регистрация: регистрация\n'
)


ADMIN_HELP_MESSAGE = (
    'База телефонов: просмотр входящих заявок в секторы
студенческого совета\n'
    'Подача документов: вкладка для получения материальной помощи
или активистской стипендии\n'
    'Создать новость: создание новостей для ленты бота.\n'
```

```
    'Загрузка шаблона для активистской стипендии: загрузка
шаблона для подачи заявления на активистскую стипендию.\n'

    'Загрузка шаблона для получения материальной помощи: загрузка
шаблона для подачи заявления на получение мат. помощи.\n'

    'Справка: вызов этого сообщения :).\n'

    'Новости: просмотр новостного блока.\n'

)


USER_HELP_MESSAGE = (

    'Подача документов: вкладка для получения материальной помощи
или активистской стипендии\n'

    'Справка: информация о командах бота\n'

    'Вступить в студенческий совет: отправить заявку в
студенческий совет для вступления в один из секторов\n'

    'Новости: узнать о последних новостях\n'

)

PHONE_SYMBOL_MAX = 12


EMAIL_REGEX = (

    r'([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-
z]{2,})+'

)
```

ПРИЛОЖЕНИЕ А.8

Файл enums.py

```
# coding: utf-8

from enum import Enum

class DocumentDestination(str, Enum):

    """Перечисление назначения документа."""

    grant = 'Получение активистской стипендии'

    material_aid = 'Получение материальной помощи'

    template_grant = 'Шаблон для активистской стипендии'

    template_material_aid = 'Шаблон для получения материальной
помощи'

class SectorName(str, Enum):

    """Названия секторов."""

    sport = 'Спортивный сектор'

    cultural = 'Культмассовый сектор'

    decorator = 'Сектор Декораторов'

    vocal = 'Вокальный сектор'

    volunteer = 'Волонтёрский сектор'

    design_creativity = 'Центр проектного творчества'

    head = 'Сектор старост'

    chairman_trade_union = 'Председатель Профсоюза'

    chairman_student_council = 'Председатель ССС'


    @staticmethod

    def get_choices():
```

```
        return [item.value for item in SectorName]



class RoleUsers(str, Enum):
    """Роли пользователя."""


    admin = 'Администратор'
    user = 'Пользователь'
```

ПРИЛОЖЕНИЕ А.9

Файл main.py

```
# coding utf-8
from aiogram.utils import executor


from handlers import admin, client
from start_bot import dp



async def on_startup(_):
    print('Бот вышел в онлайн.')



client.client_register_handlers(dispatcher=dp)
```

```
admin.admin_register_handlers(dispatcher=dp)




if __name__ == '__main__':

    executor.start_polling(dp, skip_updates=False,
on_startup=on_startup)
```

ПРИЛОЖЕНИЕ А.10

Файл models.py

```
# coding: utf-8

"""Модуль описания таблиц базы данных."""

import datetime



from sqlalchemy import (

    Column,

    DateTime,

    ForeignKey,

    Integer,

    LargeBinary,

    String,

)

from sqlalchemy.orm import declarative_base



from constans import PHONE_SYMBOL_MAX
```

```
Base = declarative_base()



class User(Base):

    """Пользователь."""


    __tablename__ = 'user'


    id = Column(Integer, primary_key=True)

    tg_user_id = Column(Integer, nullable=False)

    fio = Column(String(60), nullable=False)

    phone = Column(String(PHONE_SYMBOL_MAX), nullable=False,
unique=True)

    email_address = Column(String(30), nullable=False,
unique=True)

    status = Column(String(40), nullable=False)

    sector_name = Column(String(40), nullable=True)



class News(Base):

    """Новости."""
```

```
    __tablename__ = 'news'

    id = Column(Integer, primary_key=True)
    creator = Column(Integer, nullable=False)
    header = Column(String(30), nullable=False)
    content = Column(String, nullable=False)
    created = Column(
        DateTime(timezone=True),
        default=datetime.datetime.now(),
    )


class StudentCouncilApplication(Base):
    """Заявка в студенческий совет."""

    __tablename__ = 'student_council_application'

    id = Column(Integer, primary_key=True)
    sector_name = Column(String(40), nullable=False)
    skill = Column(String, nullable=False)
    new_skill = Column(String, nullable=False)
    tg_user_id = Column(Integer, nullable=False)
```

```
    created = Column(
        DateTime(timezone=True),
        default=datetime.datetime.now(),
    )


class Documents(Base):
    """Документы."""

    __tablename__ = 'document'

    id = Column(Integer, primary_key=True)
    document = Column(LargeBinary, nullable=False)
    file_name = Column(String, nullable=False)
    owner = Column(Integer, nullable=False)
    destination = Column(String, nullable=False)


class MaterialAidApplication(Base):
    """Заявка на получение материальной помощи."""

    __tablename__ = 'material_aid_application'
```

```
    id = Column(Integer, primary_key=True)
    tg_user_id = Column(Integer, nullable=False)
    created = Column(
        DateTime(timezone=True),
        default=datetime.datetime.now(),
    )
    document_id = Column(Integer, ForeignKey('document.id'))


class ActivistGrantApplication(Base):
    """Заявка на получение активистской стипендии."""

    __tablename__ = 'activist_grant_application'

    id = Column(Integer, primary_key=True)
    tg_user_id = Column(Integer, nullable=False)
    created = Column(
        DateTime(timezone=True),
        default=datetime.datetime.now(),
    )
    document_id = Column(Integer, ForeignKey('document.id'))
```

ПРИЛОЖЕНИЕ А.11

Файл start_bot.py

```
# coding: utf-8

from aiogram import Bot, Dispatcher

from aiogram.contrib.fsm_storage.memory import MemoryStorage

from sqlalchemy.ext.asyncio import AsyncSession,
create_async_engine


from config import Settings


memory = MemoryStorage()

config = Settings()


bot = Bot(token=config.token)

dp = Dispatcher(bot, storage=memory)

engine = create_async_engine('sqlite+aiosqlite:///psu_bot',
future=True)

async_session = AsyncSession(engine, expire_on_commit=False)
```

ПРИЛОЖЕНИЕ А.12

Файл utilits.py

```
# coding: utf-8

"""Модуль вспомогательных функций."""

import os
```

```
import smtplib

from email.mime.application import MIMEApplication

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText


from aiogram import types

from aiogram.types import ReplyKeyboardRemove,
ReplyKeyboardMarkup

from sqlalchemy import select


from constans import PHONE_SYMBOL_MAX

from enums import RoleUsers, SectorName

from keyboards import user_keyboard

from keyboards.ui import admin_keyboard

from models import User

from start_bot import async_session, bot, config



async def is_admin(user_id: int, connection) -> bool:

    """Возвращает True, если пользователь администратор.

    :param user_id: уникальный идентификатор пользователя
телеграм,

    :param connection: подключение к базе данных.
```

```
    """
    select_stmt = select(
        User.tg_user_id
    ).where(
        User.tg_user_id == user_id,
        User.status == RoleUsers.admin.value
    )
    admin_id = (await connection.execute(select_stmt)).first()
    return bool(admin_id)



async def check_sector_name(sector_name: str) -> bool:
    """Проверка ввода сектора."""
    return bool(sector_name in SectorName.get_choices())



async def phone_validator(phone_number: str) -> tuple:
    """Валидация номера телефона."""
    is_valid = True
    help_message = ''
    if len(phone_number) > PHONE_SYMBOL_MAX:
        is_valid = False
```

```
        help_message = 'Неверная длина номера.\n'
    if not phone_number.strip('+').isnumeric():
        is_valid = False
        help_message += 'Обнаружены недопустимые символы.\n'
    return is_valid, help_message



async def client_cancel(message: types.message) -> bool:
    """Кнопка отмены для пользователя."""
    is_not_cancel = True
    keyboard = ReplyKeyboardRemove()
    if message.text == 'Отмена':
        stmt_user = select(
            User.id
        ).where(
            User.tg_user_id == message.from_user.id,
            User.status == RoleUsers.user.value,
        )
        user_id = (await
async_session.execute(stmt_user)).first()
        if user_id:
            keyboard = user_keyboard
        await bot.send_message(
```

```
            message.from_user.id,
            'Возвращаю в главное меню.',
            reply_markup=keyboard,
        )
        is_not_cancel = False
    return is_not_cancel


async def admin_cancel(message: types.message) -> bool:
    """Кнопка отмены для адмика."""
    is_not_cancel = True
    keyboard = ReplyKeyboardRemove()
    if message.text == 'Отмена':
        stmt_admin = select(
            User.tg_user_id
        ).where(
            User.tg_user_id == message.from_user.id,
            User.status == RoleUsers.admin.value
        )
        admin_id = (await
async_session.execute(stmt_admin)).first()
        if admin_id:
            keyboard = admin_keyboard
```

```
        await bot.send_message(
            message.from_user.id,
            'Возвращаю в главное меню.',
            reply_markup=keyboard,
        )
        is_not_cancel = False
    return is_not_cancel


async def convert_into_bytes(file_path: str) -> bytes:
    """Конвертация файла в байты."""
    with open(file_path, 'rb') as file:
        binary = file.read()
    return binary


async def write_to_file(binary_data: bytes, file_name: str) ->
None:
    """Конвертация байтов в файл."""
    with open(file_name, 'wb') as file:
        file.write(binary_data)
```

```
async def get_keyboard(user_id: int, connection) ->
ReplyKeyboardMarkup:
    """Получение клавиатуры в общих методах."""
    keyboard = user_keyboard
    stmt_status = select(
        User.status
    ).where(
        User.tg_user_id == user_id,
    )
    status = (await connection.execute(stmt_status)).first()
    if status[0] == RoleUsers.admin.value:
        keyboard = admin_keyboard
    return keyboard



async def send_mail(
        theme_mail: str,
        message_content: str,
        to_addr: str,
        files_paths=None,
) -> None:
    """Отправка письма на почту."""
    server = smtplib.SMTP('smtp.gmail.com', 587)
```

```
server.starttls()
server.login(config.email, config.mail_password)
msg = MIMEMultipart()
# Содержимое письма.
if message_content:
    msg.attach(MIMEText(message_content, 'plain'))
# Тема письма.
msg['Subject'] = theme_mail
if files_paths:
    for file_path in files_paths:
        with open(file_path, 'rb') as file:
            # Отсекаем пути, получаем только имя файла.
            head, filename = os.path.split(file_path)
            part = MIMEApplication(
                file.read(),
                Name=filename,
            )
            # Закрепляем документ в письме.
            part['Content-Disposition'] = (
                f'attachment; filename={filename}'
            )
            msg.attach(part)
```

```
    server.sendmail(config.email, to_addr, msg=msg.as_string())
```

server.quit()

ПРИЛОЖЕНИЕ А.13

Файл aihandlers.py

```
import enum
import pandas as pd
from aiogram import Dispatcher, types
from openai import OpenAI
from config import Settings
class Prompts(enum.Enum):
    """Промты"""
    MSG_PROMPT = ("Ответь на это, учитывая эти правила:"
                  " Постарайся ответить раскрыто и без воды."
                  " Используй для ответа файл file-
YRxr3nAMkMJ8fPIGCL43qEPO."
                  " Абсолютно все цифры пиши текстом, а не
цифрами,"
                  " например вместо 1380 пиши тысяча триста
восемьдесят."
                  " Постарайся ответить простым языком,"
                  " как будто ты голосовой ассистент."
                  " Отвечай так, как будто ты девушка.
Обращайся на ты."
                  " Только если спросят твоё имя, говори, что
тебя зовут Bubba."
                  " Только если спросят где тебя сделали,"
                  " то тебя придумал Максим Митрофанов"
                  )
```

```
    class MyOpenAI:
        """Класс для взаимодействия с OpenAI"""

        def __init__(self):
            self.client = OpenAI(api_key=Settings.OPENAI_TOKEN)
            self.csv_file                                          =
"/home/williamcano/PycharmProjects/work/diploma/cwurData.csv"

        async def text_answer_gpt(self, message: str):
            """Ответ текстом от ГПТ"""
            completion = self.client.chat.completions.create(
                model="gpt-4o",
                messages=[
                    {"role":         "system",         "content":
Prompts.MSG_PROMPT.value},
                    {"role": "user", "content": message}
                ]
            )
            return completion.choices[0].message.content

        def upload_file(self) -> str:
            """Загрузка файлов в OPenAI"""
            with open(self.csv_file, "rb") as binary_file:
                upload_file = self.client.files.create(
                    file=binary_file,
                    purpose="assistants"
                )
                return upload_file.id


    async def ai_msg_response(message: types.Message):
```

```
        """метод для обработки текста исходя из
         сообщения со вставкой промта и датасета"""
        if message.text:
            response = await
MyOpenAI().text_answer_gpt(message.text)
            await message.answer(response)


    def ai_handlers(dispatcher: Dispatcher):
        """Регистрация всех хендлеров"""
        dispatcher.register_message_handler(
            callback=ai_msg_response
        )
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Интеллектуальная диалоговая система вуза

Презентация

Интеллектуальная диалоговая система вуза

Автор работы: студент группы 22ВВИМ1 М.П. Митрофанов, ПГУ
Научный руководитель: к.т.н., доцент кафедры ВТ
Карамышева Н.С.

SLIDESMANIA.COM

Цель магистерской диссертации

Целью работы является освоение актуальных и современных моделей искусственного интеллекта, способов и методов обучения искусственного интеллекта, средств программной разработки; получение навыков разработки чат-диалоговых информационных систем на базе мессенджеров; реализация чат-бота, необходимого для автоматизации работы совета студенческого самоуправления и мгновенной поддержки студентов по учебным вопросам.

SLIDESMANIA.COM

Аналоги

- Чат-бот Томского Политехнического Университета
- Чат-бот ИАТЭ
- Чат-боты файлообменники
- Сайт ПГУ
- Посещение деканата

SLIDESMANIA.COM

Стек технологий

Язык программирования

PostgreSQL
Система управления базами данных

Среда разработки

Мессенджер

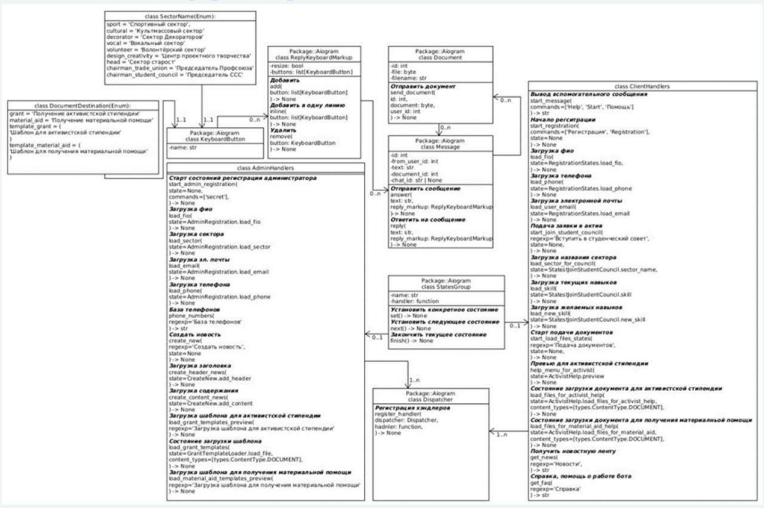
ChatGPT
ИИ-ассистент

SQLAlchemy
Библиотека

Библиотека

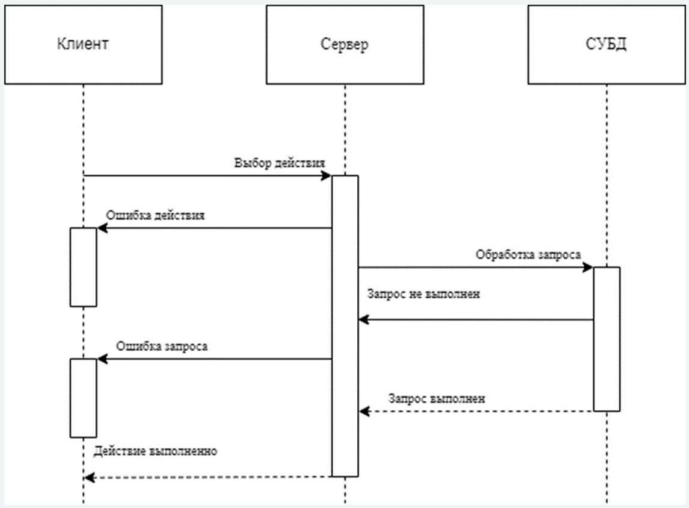
SLIDESMANIA.COM

Диаграмма классов



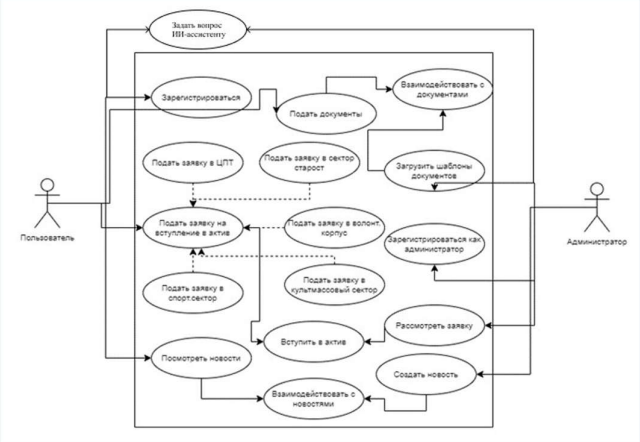
SLIDESMANIA.COM

Диаграмма последовательности



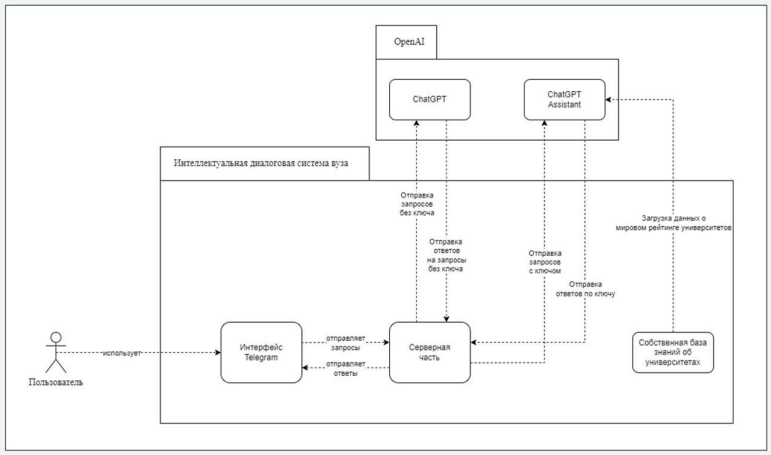
SLIDESMANIA.COM

Диаграмма прецедентов



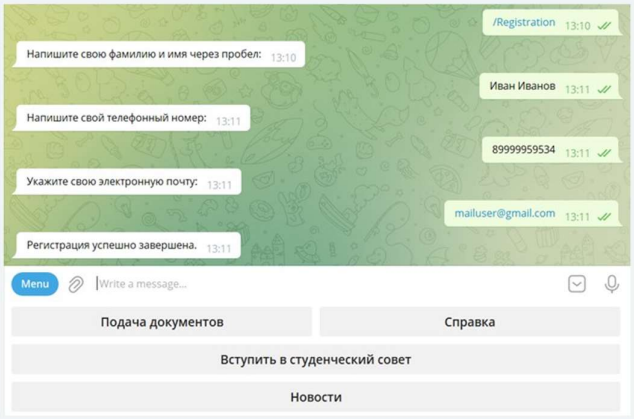
SLIDESMANIA.COM

Архитектура системы



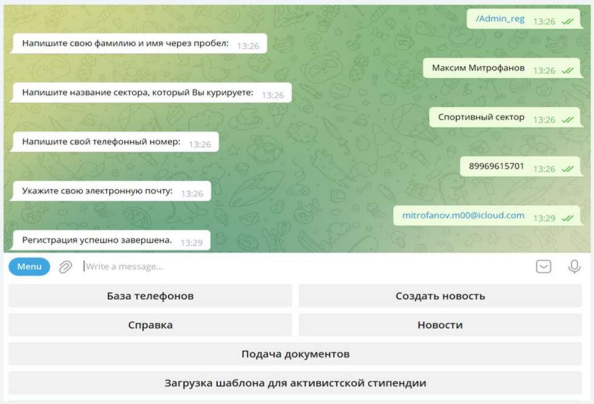
SLIDESMANIA.COM

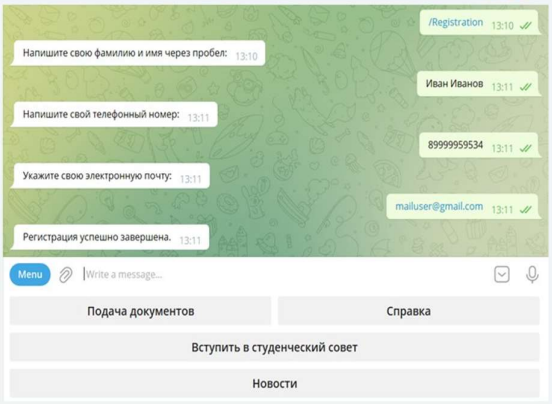
Регистрация пользователей



SLIDESMANIA.COM

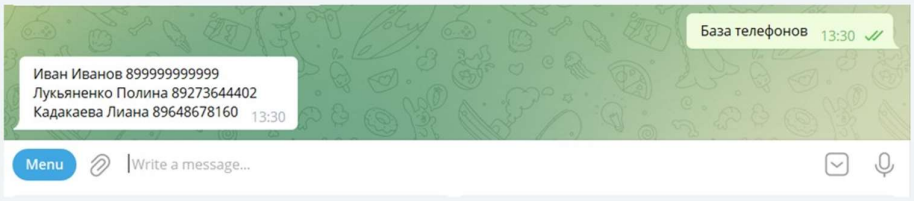
Регистрация пользователей





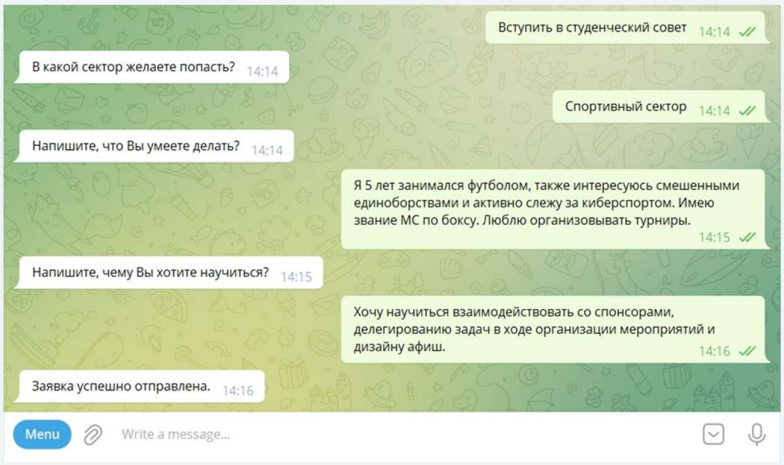
SLIDESMANIA.COM

Просмотр базы телефонов



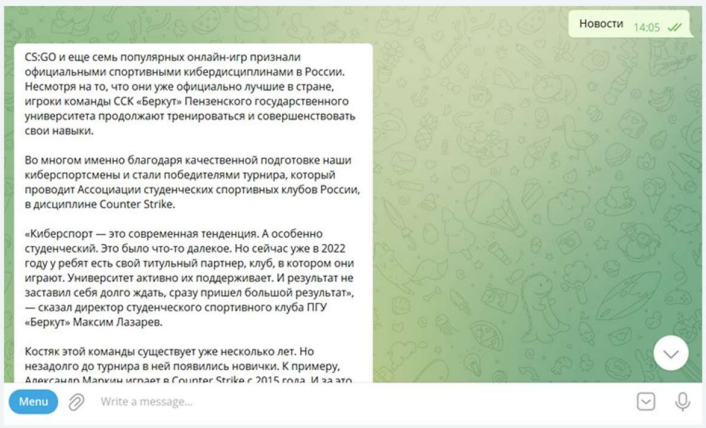
SLIDESMANIA.COM

Подача заявки на вступление в ССС



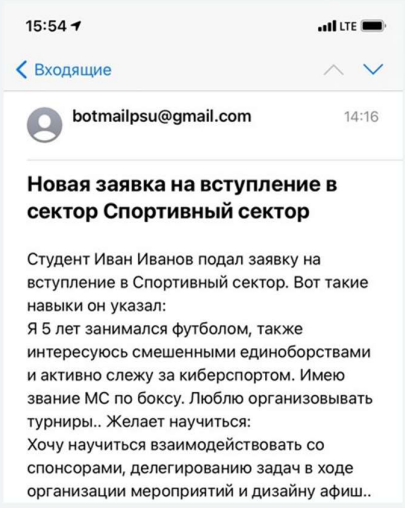
SLIDESMANIA.COM

Раздел новости



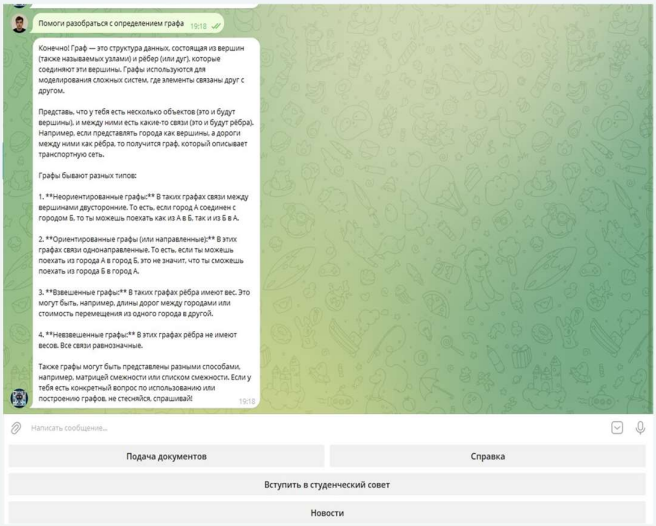
SLIDESMANIA.COM

Почтовое оповещение



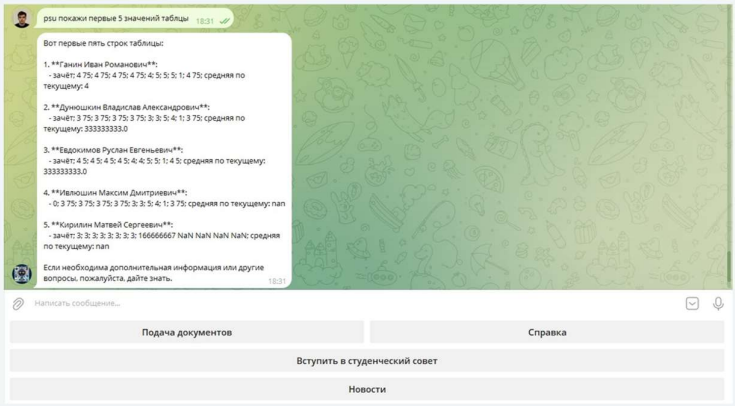
SLIDESMANIA.COM

Работа ИИ-ассистента



SLIDESMANIA.COM

Работа ИИ-ассистента по загруженному файлу



SLIDESMANIA.COM

Заключение

В ходе разработки проекта изучены методы интеграции ИИ-ассистентов и способы загрузки датасетов для них

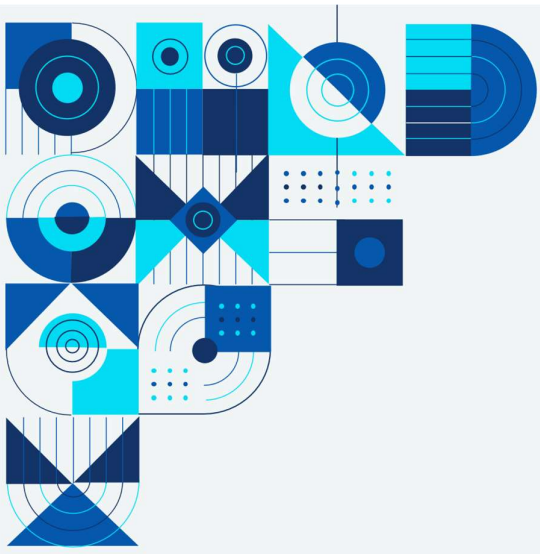
Был проведен анализ действующих аналогов чат-ботов, а также их плюсы и минусы

В ходе реализации проекта были глубоко изучены Python, SQLite, Aiogram, OpenAI ChatGPT

Были разработаны архитектура бота, алгоритм загрузки датасета

Разработан готовый к эксплуатации в новом учебном году чат-бот для Совета студенческого самоуправления

SLIDESMANIA.COM



Спасибо за внимание!

SLIDESMANIA.COM