Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ЛЕСОСИБИРСКИЙ ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ –
филиал Сибирского федерального университета

Высшей математики, информатики, экономики и естествознания
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
Л.Н. Храмова
подпись инициалы, фамилия
«11» июня 2025 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.02 Информационные системы и технологии
код-наименование направления

РАЗРАБОТКА ЧАТ-БОТА ДЛЯ ИНФОРМАЦИОННОГО СОПРОВОЖДЕНИЯ ПРИЕМНОЙ КАМПАНИИ ЛПИ – ФИЛИАЛА СФУ

Руководитель 11.06.2025 доцент, канд. пед. наук Е.В. Киргизова
подпись, дата; должность, ученая степень; инициалы, фамилия

Выпускник 11.06.2025 С.А. Пасканный
подпись, дата; инициалы, фамилия

Нормоконтролер 11.06.2025 Е.В. Киргизова
подпись, дата; инициалы, фамилия

Лесосибирск 2025

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка чат-бота для информационного сопровождения приёмной кампании ЛПИ – филиала СФУ» содержит 80 страниц текстового документа, 31 иллюстрацию, 4 таблицы, 40 использованных источников.

PYTHON, TELEGRAM BOT API, ЧАТ-БОТ, ИНФОРМАЦИОННОЕ СОПРОВОЖДЕНИЕ, ПРИЁМНАЯ КАМПАНИЯ, АЛГОРИТМЫ, UX-ДИЗАЙН, АВТОМАТИЗАЦИЯ.

Цель исследования – теоретически обосновать и разработать чат-бот для информационного сопровождения приемной кампании ЛПИ – филиала СФУ с использованием языка программирования Python.

Объект исследования – чат-бот для приёмной кампании.

Предмет исследования – процесс разработки чат-бота для информационного сопровождения приёмной кампании.

В процессе работы были поставлены и решены следующие задачи:

– изучить существующие решения и подходы к разработке чат-ботов для информационного сопровождение приемной кампании;

– выявить потребности целевой аудитории (студентов, абитуриентов, сотрудников);

– обосновать выбор программных средств разработки чат-бота для информационного сопровождение приемной кампании;

– разработать архитектуру и структуру чат-бота приемной кампании;

– провести тестирование работоспособности чат-бота для информационного сопровождение приемной кампании.

Разработанный чат-бот обеспечивает автоматическую выдачу актуальной информации по ключевым вопросам приёмной кампании, сокращая нагрузку на сотрудников и повышая доступность информации.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Быстрое принятие решений, выполнение нескольких задач одновременно и потребность в минимизации рисков создают необходимость в автоматизации бизнес-процессов. Автоматизация бизнеса представляет собой передачу части операций под контроль информационной системы [1]. Развитие цифровых технологий оказывает значительное влияние на все сферы жизни современного общества, включая сферу образования. Сегодня образовательные учреждения стремятся внедрять инновационные решения для повышения качества обслуживания студентов, абитуриентов, преподавателей и других заинтересованных лиц. Одним из основных направлений в этой области является использование чат-ботов – программных решений, позволяющих автоматизировать общение между пользователями и информационными системами.

Чат-боты находят широкое применение в самых разных отраслях: от коммерции до здравоохранения, от туристических сервисов до государственной службы. В условиях постоянного роста потока обращений пользователей и увеличения количества информации, которую необходимо оперативно передавать, чат-боты выполняют большое количество задач, но в основном их главная задача, это, снижение нагрузки на операторов.

Актуальность темы исследования обусловлена тем что, пользователи нередко сталкиваются с трудностями в поиске нужной информации, особенно в период приёмной кампании. Создание чат-бота, способного автоматически предоставлять актуальные ответы на часто задаваемые вопросы, позволит повысить доступность и оперативность обслуживания.

Цель исследования – теоретически обосновать и разработать чат-бот для информационного сопровождения приемной кампании ЛПИ – филиала СФУ с использованием языка программирования Python.

Объект исследования – чат-бот для приемной кампании.

Предмет исследования – процесс разработки чат-бота для информационного сопровождение приемной кампании.

Для достижения поставленной цели необходимо решить следующие задачи:

– изучить существующие решения и подходы к разработке чат-ботов для информационного сопровождение приемной кампании;

– выявить потребности целевой аудитории (студентов, абитуриентов, сотрудников);

– обосновать выбор программных средств разработки чат-бота для информационного сопровождение приемной кампании;

– разработать архитектуру и структуру чат-бота приемной кампании;

– провести тестирование работоспособности чат-бота для информационного сопровождение приемной кампании.

Методы исследования:

1. теоретические: анализ научной и учебной литературы по информационным системам и чат-ботам, сравнительный анализ существующих чат-ботов для информационного сопровождения, моделирование взаимодействия пользователя с системой с использованием методологии IDEF0, проектирование архитектуры чат-бота и базы данных.

2. эмпирические: наблюдение за процессом информационного сопровождения приёмной кампании во время практики, разработка и тестирование прототипа чат-бота, функциональное и модульное тестирование, отладка и проверка корректности работы чат-бота.

В процессе разработки чат-бота для информационного сопровождения приёмной кампании были использованы современные средства программирования: язык Python, библиотека для работы с Telegram Bot API (или укажите свою), а также встроенная в код структура хранения данных без использования внешней базы данных. Такой выбор технологий обусловлен требованиями к простоте реализации, автономности работы и минимизации зависимостей при установке и запуске конечного продукта.

Результаты исследования представлены на следующих научных мероприятиях:

1. VIII Всероссийская научно-практическая конференция «Актуальные проблемы преподавания дисциплин естественнонаучного цикла», ЛПИ – филиал СФУ (сертификат участника).

2. IV Всероссийской научно-практической конференции «Современное педагогическое образование: теоретический и прикладной аспекты» в секции «Информатика, информационные технологии и экономика», ЛПИ – филиал СФУ (сертификат участника).

По результатам исследования опубликована статья в сборнике конференции:

1. Пасканный, С.А. Использование чат-бота для работы с сайтом организации / С. А. Пасканный // Актуальные проблемы преподавания дисциплин естественнонаучного цикла, Тезисы докладов VIII Всероссийская научно-практическая конференция преподавателей, учителей, студентов и молодых ученых, ЛПИ – филиал СФУ – Лесосибирск, 2024.

Структура выпускной квалификационной работы: выпускная квалификационная работа состоит из введения, двух глав, заключения, 2 приложений и списка использованных источников, включающего 40 наименований.

1 Постановка задачи на проектирование чат-бота для информационного сопровождения приемной кампании

1.1 Особенности чат-бота: функциональные и архитектурные

Чичулин А. В. понятие чат-бот вводит следующим образом «это компьютерная программа, которая предназначена для имитации человеческого общения с помощью обмена сообщениями или голосовых интерфейсов» [37].

Чат-боты – это специальные программы, которые автоматически помогают пользователям общаться с информационными системами. В образовательной сфере, особенно во время приёмной кампании, чат-боты используются для оперативного предоставления справочной информации, обработки типовых запросов и оптимизации процесса коммуникации с абитуриентами.

Рассмотрим функциональные особенности, предъявляемые к чат-ботам.

С точки зрения функциональности чат-боты условно подразделяются на два основных типа:

– Сценарные чат-боты (rule-based) – функционируют на основе заранее запрограммированных сценариев, где пользовательский ввод распознаётся с помощью ключевых слов или выбора кнопок. Такие системы не обладают способностью к обучению или генерации новых ответов, что ограничивает их гибкость. Для обновления информации или изменения логики взаимодействия требуется ручное вмешательство разработчиков. Сценарные чат-боты используются в различных сферах, например:

– Клиентская поддержка. Боты помогают быстро отвечать на вопросы, обрабатывать запросы и предоставлять информацию. Чат-бот поддержки представлен на рисунке 1.

– Электронная коммерция. В интернет-магазинах боты помогают выбирать товары, оформлять заказы, отслеживать доставку и отвечать на часто задаваемые вопросы.

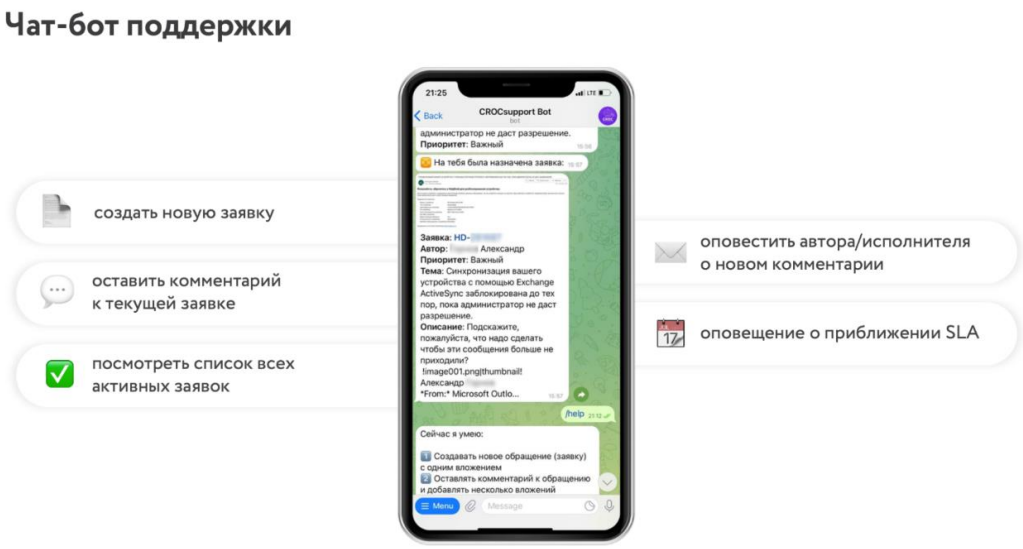

Рисунок 1 – Чат-бот клиентской поддержки

– Маркетинг и продажи. Чат-боты используются для взаимодействия с потенциальными клиентами, проведения опросов, сбора данных и отправки персонализированных предложений.

– HR и рекрутинг. В отделах кадров боты автоматизируют процесс отбора кандидатов, помогают проводить первичные собеседования. Чат-бот рекрутинга представлен на рисунке 2.

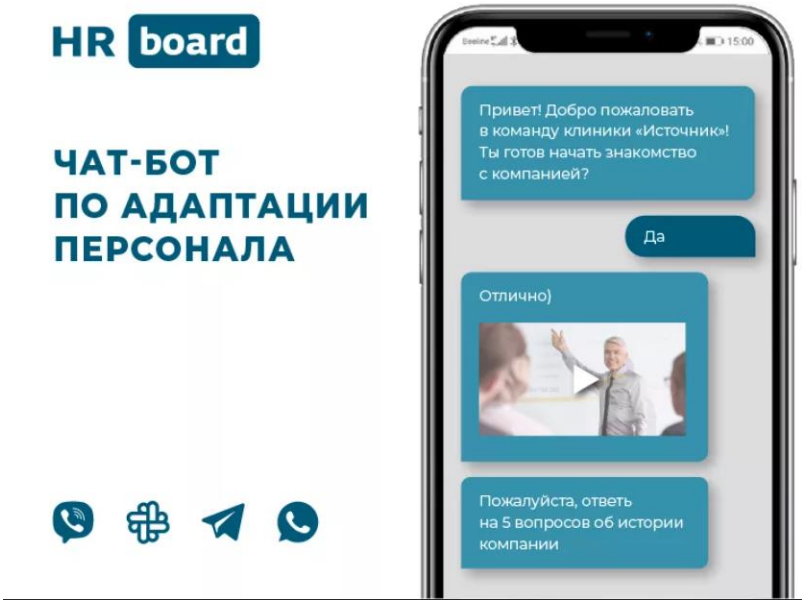

Рисунок 2 – Чат-бот HR и рекрутинга

– Банковская и финансовая сфера. Боты в банках помогают клиентам проверять баланс, переводить деньги, оплачивать счета и получать консультации по продуктам.

– Интеллектуальные чат-боты (AI-powered) – используют технологии обработки естественного языка (NLP), машинного обучения и, при наличии ресурсов, глубокого обучения для интерпретации произвольных пользовательских запросов. Эти системы способны на адаптацию и обучение на новых данных, что позволяет им обеспечивать более естественное и человечное взаимодействие.

В рамках исследования отказ от использования AI (Artificial Intelligence, искусственный интеллект) обусловлен необходимостью минимизировать ресурсы на поддержку и обеспечить стабильную работу при ограниченных вычислительных мощностях. Поэтому основная логика чат-бота для приёмной кампании строится на сценарных моделях с ключевыми функциями базовой обработки текста (например, регулярными выражениями):

– автоматизированное консультирование по основным вопросам приёма – правила, документы, сроки;

– информирование об актуальных мероприятиях, датах подачи документов и результатах экзаменов;

– маршрутизация запросов – выбор ответственного отдела или специалиста на основе типа вопроса;

– перенаправление сложных или нестандартных вопросов на живого оператора;

– сбор и ведение статистики обращений, анализ популярных запросов для оптимизации базы знаний.

Исследование Гюльдала и Динчера демонстрирует, что студенты высоко оценивают сценарные чат-боты в образовательной сфере: такие системы обеспечивают качественные ответы, быстрый доступ к информации и простоту

эксплуатации, даже несмотря на отсутствие AI-модулей, и при этом требуют значительно меньших ресурсов для поддержки [11].

Выделим архитектурные особенности проектирования чат-бота. Архитектура чат-бота проектируется по модульному принципу, что позволяет обеспечить масштабируемость и гибкость системы. Структура чат-бота зависит от предъявляемых требований и используемых технологий при разработке.

На рисунке 3 представлена современная архитектура чат-бота, которая базируется на нескольких фундаментальных компонентах.

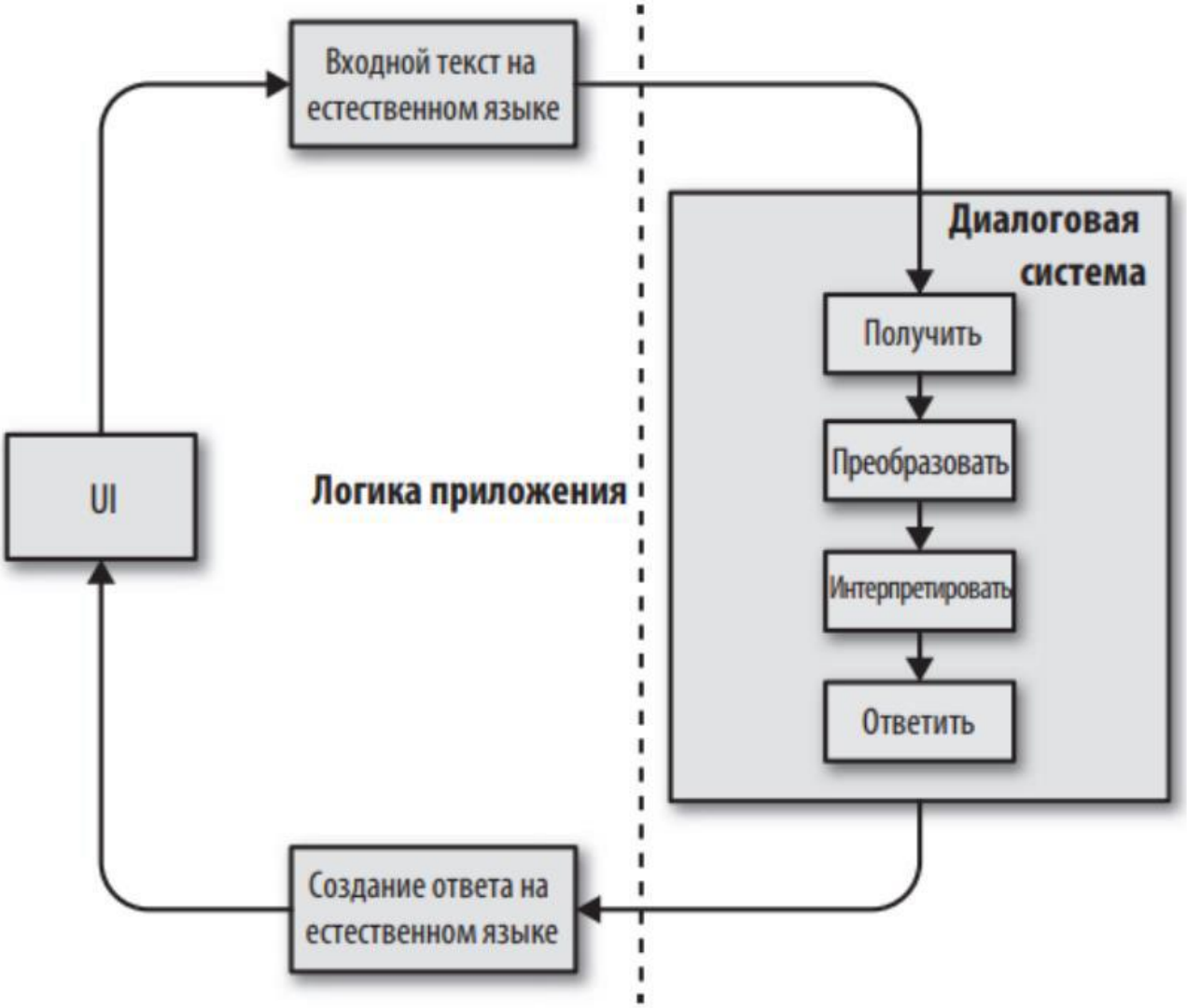

Рисунок 3 – Архитектура чат-бота

Первый компонент – интерфейс с пользователем, осуществляющий прием ввода пользователя (ввод речи с помощью микрофонов или ввод текста через

веб-интерфейс) и передачу интерпретируемого вывода (через динамики для вывода синтезированной речи или экран мобильного устройства для вывода текста). Второй компонент – внутренняя диалоговая система, которая интерпретирует входных данные, управляет внутренним состоянием и генерирует ответы. Во внутренней системе можно выделить 4 главных элемента:

– получение входящих данных от пользователя (текст или речь);

– понимание входного сообщения (NLU);

– управление диалогом (принятие решения о дальнейшем диалоге);

– генерация ответа (NLG).

На основании рисунка 3 выделим основные компоненты разрабатываемого чат-бота:

– интерфейс взаимодействия – интеграция с мессенджерами (Telegram), веб-виджет, обеспечивающий доступ с сайта университета;

– обработчик диалога – центральный модуль управления логикой и состоянием диалогов, реализующий сценарии и управление сессиями;

– обработчик диалога (модуль управления логикой);

– база знаний и справочные данные – хранение часто задаваемых вопросов, документов, расписаний и регламентов;

– службы интеграции – API (интерфейс программирования приложений) университета (например, данные о расписании и статусе документов).

Подход к архитектуре чат-ботов, описанный выше, соответствует типовой модели, представленной в ряде исследований, где рассматриваются как сценарные, так и интеллектуальные решения.

Таким образом, такой подход позволяет расширить функционал чат-бота, подключать более интеллектуальные модули или дополнительные каналы коммуникации.

1.2 Требования к программному обеспечению чат-бота

Требования к разрабатываемому программному обеспечению можно разделить на: функциональные, нефункциональные и организационные.

1. Функциональные требования:

– обеспечение быстрого и корректного ответа на наиболее распространённые вопросы абитуриентов, касающиеся правил приёма, сроков и документов;

– обработка запросов с элементами анализа текста для определения направления и маршрутизации пользователя, без использования глубокого NLP;

– ведение истории диалогов для поддержки контекста в рамках одной сессии, что позволяет отвечать последовательно и адекватно;

– поддержка основных коммуникационных платформ – Telegram и веб-сайт университета.

2. Организационные требования:

– возможность оперативного редактирования базы знаний без необходимости полного перекомпилирования или деплоя проекта, например, за счёт хранения данных в отдельном конфигурационном файле (JSON), доступном для редактирования.

– документированная структура базы данных, хранящейся в коде (в том числе формат вопросов, ответов, сценариев, маршрутов), чтобы сотрудники могли корректно вносить изменения.

– наличие шаблонов сценариев для добавления новых вопросов/ответов, структурированных по направлениям обучения.

– логика fallback-ответов – если введённый вопрос не найден в базе, бот должен предлагать варианты или передавать на оператора, чтобы не приводить к «тупикам».

Современные исследования Чичулина А.В. и Джанарсанам С. подтверждают, что интеграция чат-ботов с цифровыми платформами

университета и возможность оперативного редактирования контента особенно важны в период приёмной кампании, когда нагрузка на службы поддержки возрастает [37, 12].

Вышеизложенное позволяет сделать вывод о том, что разработка чат-бота для поддержки абитуриентов требует чёткого соблюдения как функциональных, так и организационных требований. Функциональные аспекты обеспечивают быстрый, точный и контекстно корректный ответ на запросы пользователей, а организационные – гибкость и удобство администрирования базы знаний без остановки работы системы. Выделенные требования обеспечат эффективность взаимодействия с абитуриентами и улучшает качество предоставляемой информации

1.3 Сравнительный анализ аналогов разрабатываемого чат-бота

Для разработки эффективного и функционального чат-бота, сопровождающего приемную кампанию, необходимо проанализировать существующие решения. В ходе исследования изучено несколько чат-ботов ведущих российских университетов, реализованных на различных платформах и использующих разные технологические подходы.

Основные параметры анализа:

– технологическая база: Используемые платформы и фреймворки, наличие или отсутствие искусственного интеллекта, степень автоматизации;

– интеграция с внутренними системами: Связь с университетскими информационными системами (расписания, CRM, базы данных);

– платформы взаимодействия: Телеграм, ВКонтакте, веб-сайты, мобильные приложения;

– функциональные возможности: Объем предоставляемой информации, поддержка диалогов, возможность записи на консультации, маршрутизация запросов;

– пользовательский опыт: Интуитивность, скорость реакции, качество ответов;

– ограничения и проблемы: Технические, ресурсные и организационные.

Подобный подход к анализу реализованных решений и выбору платформ подтверждается в современных исследованиях по разработке чат-ботов в образовательной сфере [7].

1. Московский государственный университет (МГУ)

Использует Google Dialogflow как NLP-платформу, интегрированную с сайтом и Telegram. Бот способен распознавать естественный язык, что улучшает качество ответов и позволяет обрабатывать сложные запросы. Однако для оптимальной работы требует регулярной доработки моделей и наличия специалистов по AI. Высокая сложность поддержки ограничивает масштабируемость. Чат-бот Московского государственного университета (МГУ) представлен на рисунке 4.

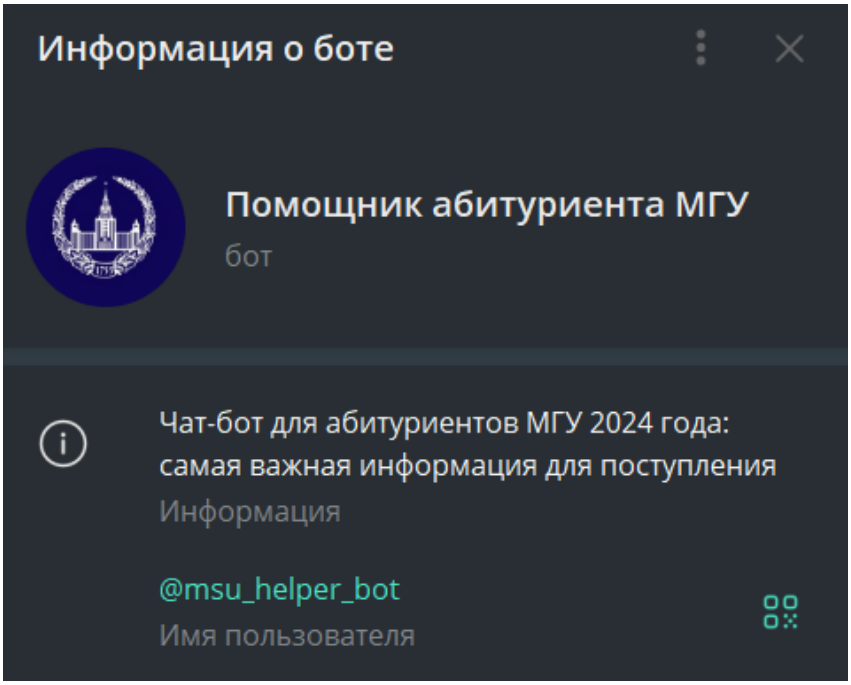

Рисунок 4 – Чат-бот приемной кампании МГУ

2. Санкт-Петербургский государственный университет (СПбГУ)

Кастомная платформа с глубоким уровнем интеграции с расписанием и личным кабинетом абитуриента. Преимущество – возможность предоставлять

актуальные данные в реальном времени. Однако бот функционирует преимущественно через сайт, ограничивая доступность и удобство для пользователей, предпочитающих мессенджеры. Чат-бот Санкт-Петербургского государственного университета (СПбГУ) представлен на рисунке 5.

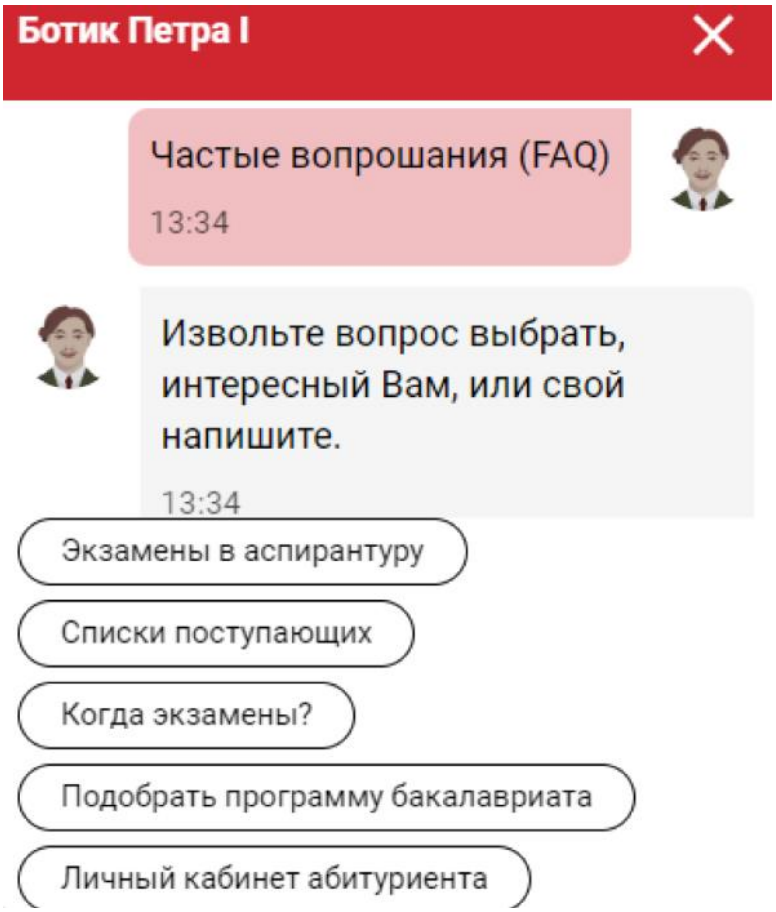

Рисунок 5 – Чат-бот приемной кампании СПбГУ

3. РАНХиГС (Российская академия народного хозяйства и государственной службы)

Использует коммерческое решение Chat2Desk, которое поддерживает мультиканальность (WhatsApp, Telegram, Viber). Система обладает продвинутой аналитикой и возможностями интеграции с CRM. Основной минус – высокая стоимость лицензирования и зависимость от внешнего провайдера, что ограничивает возможности кастомизации. Чат-бот РАНХиГС представлен на рисунке 6.

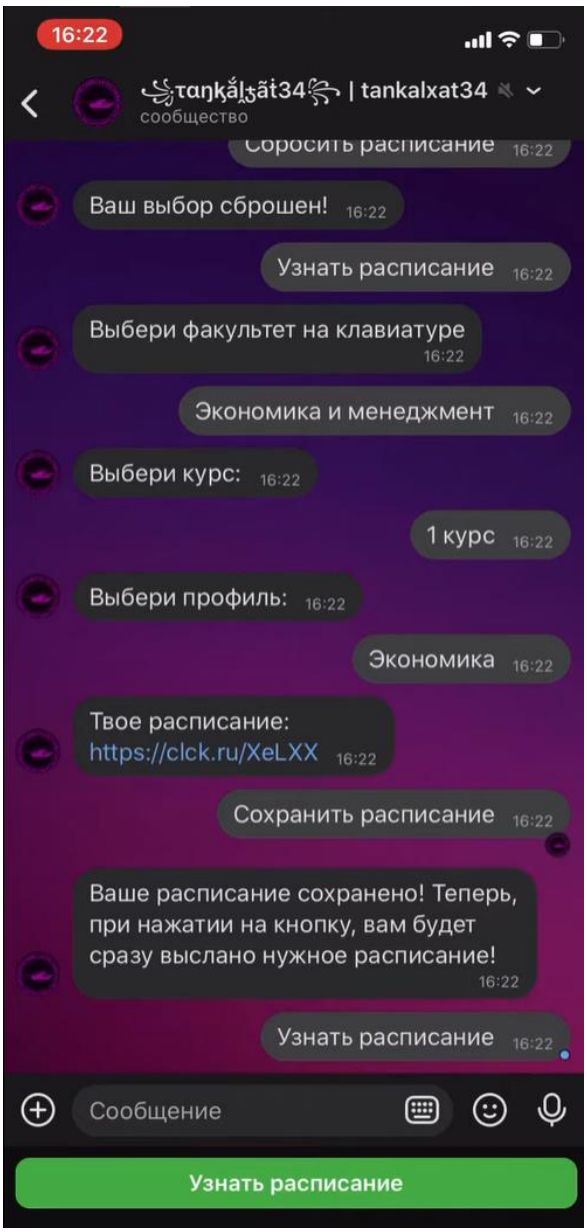

Рисунок 6 – Чат бот РАНХиГС

4. Тюменский государственный университет

В ТюмГУ используется чат-бот, разработанный с применением Telegram Bot API. Решение базируется на сценарном подходе, где обработка пользовательских запросов осуществляется через набор ключевых слов. Такая архитектура обеспечивает стабильную работу и удобство внедрения, однако не предусматривает продвинутого анализа естественного языка, что ограничивает вариативность ответов и адаптивность под нетипичные запросы. Чат-бот Тюменского государственного университета представлен на рисунке 7.

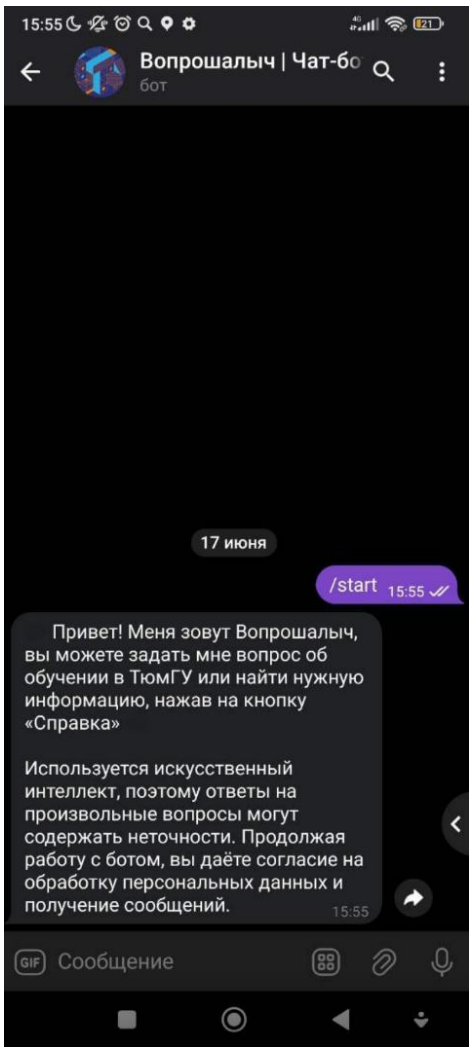

Рисунок 7 – Пример чат-бота Тюменского государственного университета

5. Казанский федеральный университет (КФУ)

Для реализации чат-бота в КФУ была выбрана платформа с открытым исходным кодом – Rasa, позволяющая создавать интеллектуальных виртуальных помощников. Такое решение отличается высокой степенью настройки и широкими возможностями в части обработки естественного языка. Однако разработка на Rasa требует квалифицированных специалистов и серьёзных вычислительных ресурсов. В некоторых ситуациях университету бывает сложно оперативно масштабировать решение из-за ограниченных ресурсов и технических сложностей. Чат-бот Казанского федерального университета (КФУ), представлен на рисунке 8.

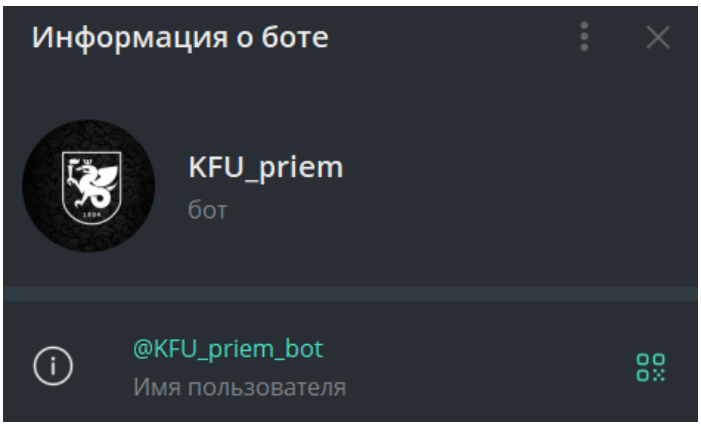

Рисунок 8 – Чат-бот приемной кампании КФУ

6. Пермский политехнический университет (Пермский политех)

Разработан прототип чат-бота на языке JavaScript, проверенный на работоспособность в группе ВКонтакте. Чат-бот Пермского политехнического университета внедрен в работу приемной кампании с июня 2023 года. Проект ориентирован на повышение оперативности обработки запросов абитуриентов и автоматизацию коммуникаций. Чат-бот Пермского Политеха, представлен на рисунке 9.

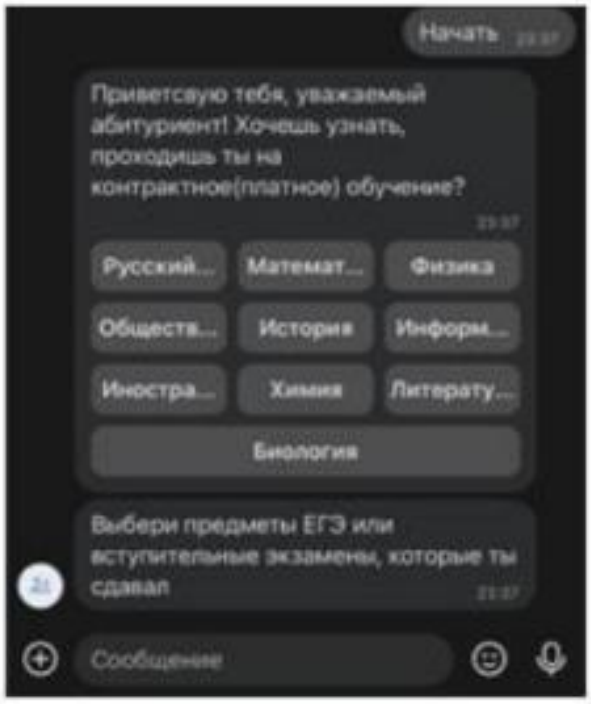

Рисунок 9 – Чат-бот приемной кампании Пермского политехнического университета

Таблица 1 – Сравнительный анализ использования чат-ботов в российских вузах

Наименование ВУЗА	Используемая технология	Платформа реализации	Преимущества	Недостатки
Московский государственный университет МГУ	Google Dialogflow	Сайт, Telegram	Поддержка естественного языка, обработка сложных запросов	Требует регулярной доработки и специалистов по AI
Санкт-Петербургский государственный университет СПбГУ	Кастомная платформа	Сайт	Интеграция с расписанием и личным кабинетом	Ограниченная доступность через мессенджеры
Российская академия народного хозяйства и государственной службы РАНХиГС	Chat2Desk	WhatsApp, Telegram, Viber	Мультиканальность, аналитика, интеграция с CRM	Высокая стоимость, ограниченная кастомизация
Тюменский госуниверситет	Telegram Bot API	Telegram	Простота реализации, высокая надёжность	Отсутствие NLP, ограниченные сценарии
Казанский федеральный университет КФУ	Rasa	Сайт, Telegram	Гибкость, open-source, настройка NLP	Требует обученной команды, ресурсоёмкость
Пермский Политех	JavaScript	ВКонтакте	Быстрая проверка прототипа, договорённость о внедрении	Ограниченная функциональность прототипа, узкая платформа

На основании сравнительного анализа представленного в таблице 1 выделим следующие недостатки:

– отсутствие единой универсальной модели. В зависимости от задач, бюджета и технических возможностей вузы выбирают либо готовые коммерческие решения, либо создают кастомные боты с минимальным AI.

– проблемы поддержки. AI-боты требуют постоянного обучения и модификации сценариев, что зачастую становится серьёзной нагрузкой для ИТ-служб.

– ограниченная интеграция. Не все решения полноценно связаны с внутренними системами университета, что снижает качество информации и удобство пользователей.

– пользовательский опыт существенно различается: более сложные чат-боты обеспечивают расширенный функционал, но требуют значительных ресурсов для поддержки и обучения, тогда как более простые решения обычно демонстрируют большую стабильность, однако обладают ограниченной гибкостью и возможностями.

Таким образом, «в проекте предлагается создать сценарный чат-бот с ограниченным NLP и качественной интеграцией, с внутренними базами университета, что позволит минимизировать сложности поддержки и обеспечить высокий уровень информативности и удобства» [9].

1.4 Выбор технологий для разработки чат-бота

Выбор технологий является ключевым этапом в проектировании программного обеспечения. При разработке чат-бота особое внимание уделяется выбору платформы взаимодействия, языка программирования, фреймворка и системы управления базами данных, так как именно от этих компонентов зависит эффективность разработки, поддерживаемость и стабильность функционирования системы.

1.4.1 Выбор мессенджеров и онлайн-сервисов

Особую роль играет выбор мессенджера, поскольку именно он определяет канал коммуникации с пользователем. В условиях российской цифровой среды наибольшее распространение получили такие платформы, как WhatsApp, Telegram, ВКонтакте и Viber, каждая из которых обладает своими техническими и пользовательскими особенностями. Перед тем как выбрать конкретную платформу, был проведён сравнительный анализ, включающий в себя такие критерии, как популярность среди пользователей, наличие инструментов для создания ботов, удобство использования с точки зрения конечного пользователя, а также поддержка разработчиков.

WhatsApp – один из самых популярных мессенджеров в мире. Его официальное API доступно только для бизнес-аккаунтов, требует подтверждения и регистрации через Meta Business Manager. API предоставляет ограниченные возможности для создания ботов и автоматизации, а также не всегда доступен для открытого тестирования и разработки [38].

ВКонтакте – социальная сеть с собственным API, который позволяет создавать ботов и интеграции, ориентированные преимущественно на пользователей внутри платформы. API предоставляет широкий функционал для работы с сообщениями, пользователями и сообществами, но ограничивает охват аудитории вне VK [8].

Viber – платформа с официальным REST API для создания ботов и интеграций, включающим поддержку сообщений, кнопок и мультимедийных вложений. Несмотря на хорошие возможности API, в России Viber менее популярен, чем Telegram или WhatsApp, что ограничивает его применение [39].

Telegram – быстрорастущий мессенджер с открытым и хорошо документированным Bot API, доступным для всех разработчиков без ограничений. Telegram API поддерживает разнообразные инструменты: клавиатуры, команды, кнопки, мультимедийные вложения, интеграции с внешними сервисами, что делает его удобным для создания сложных чат-ботов и интерактивных интерфейсов. Активно набирает популярность в России с 2022 года [33].

Таблица 2 – Сравнительный анализ

Характеристика	**Telegram**	**WhatsApp**	**Вконтакте**	**Viber**
Популярность среди пользователей	очень высокая	высокая	высокая	средняя
Наличие инструментов для создания ботов	высокая	низкая	средняя	низкая
Удобство использования для пользователя	высокое	низкое	среднее	низкое
Поддержка разработчиков (документация, API)	высокая	низкая	средняя	средняя

По итогам сравнительного анализа представленного в таблице 4, можно заметить, что все мессенджеры отстают от быстро растущего и развивающегося

Telegram. Он выделяется на фоне остальных благодаря высокой популярности, удобству использования, широким возможностям для создания ботов и качественной поддержке разработчиков. Благодаря открытому и хорошо задокументированному API, а также наличию специализированного Bot API, Telegram предоставляет больше свободы и гибкости при разработке чат-ботов [33]. Всё это делает его оптимальной платформой для реализации проекта.

Telegram, в отличие от других платформ, предлагает собственный Bot API, благодаря которому разработчики могут с нуля создать чат-бота любой сложности. Кроме того, Telegram работает на всех популярных устройствах и не требует отдельной регистрации для запуска бота. Управление ботами происходит через специального помощника – BotFather, который позволяет за пару минут зарегистрировать нового бота и получить токен доступа к API.

Согласно исследованиям Mediascope, к 2023 году Telegram вошёл в пятёрку самых популярных интернет-ресурсов России, ежедневно им пользуются десятки миллионов человек. Всё это делает Telegram оптимальной платформой для реализации чат-бота [2].

Далее рассмотрим основные способы разработки чат-ботов на этой платформе. Для создания чат-ботов существует два основных подхода:

– без программирования (no-code/low-code) – с использованием специальных конструкторов;

– с программированием (код) – разработка логики вручную на одном из языков программирования.

Ноукод-платформы

Сегодня существует множество онлайн-сервисов для быстрого создания ботов: Manybot [23], BotFather [5], BotTap [6], PuzzleBot [27], Chatfuel [4] и другие. Эти инструменты позволяют без знаний программирования собирать бота из готовых блоков, используя визуальный интерфейс. Такой подход удобен для простых задач – например, справочных ботов, автоответчиков или опросов.

Однако есть и минусы:

– ограниченный функционал – нельзя реализовать сложную логику, нестандартные сценарии;

– зависимость от платформы – если сервис перестанет работать, бот будет недоступен;

– стоимость – расширенные возможности чаще всего доступны только по подписке;

– меньший контроль над данными – все действия идут через сторонний сервер.

Разработка вручную

Если нужен максимально гибкий и функциональный бот, который сможет выполнять любые задачи, лучше всего создавать его самостоятельно, с помощью программирования. Такой подход требует определённых знаний и навыков в языках программирования, например, Python, Java или JavaScript. Однако именно ручная разработка даёт полный контроль над ботом и его возможностями.

Преимущества ручной разработки:

– гибкость. Можно реализовать любые функции, которые нужны именно вашему проекту, без ограничений, которые бывают в конструкторах.

– безопасность. Вы сами управляете хранением данных и логикой работы бота, что снижает риски, связанные с использованием сторонних платформ.

– интеграция. Бота можно подключить к другим системам, базам данных и сервисам, что расширяет его возможности.

– отсутствие регулярных платежей. В отличие от ноукод-платформ, при самостоятельной разработке не нужно платить абонентскую плату за использование сервиса – достаточно только расходов на сервер и домен, если они нужны.

Недостатки:

– требуется знание программирования и опыт разработки для такой работы.

– создание такого бота занимает больше времени по сравнению с использованием конструкторов.

– необходимость самостоятельно заботиться о хостинге и поддержке.

Таблица 3 – Сравнительный анализ подходов к созданию чат-ботов

Критерий	С кодом	Ноукод-платформа
Скорость запуска	Низкая	Высокая
Необходимость знаний	Требуется программирование	Не требуется
Гибкость, настройка логики	Полная свобода	Только то, что предусмотрено
Надёжность	Высокая (при правильной настройке)	Зависят от сторонней платформы
Стоимость	Бесплатно (кроме хостинга)	Подписка, платные функции

По итогам сравнительного анализа представленного на таблице 3 видно, что разработка чат-бота с помощью программирования требует больше времени и навыков, однако обеспечивает максимальную гибкость, надежность и отсутствие постоянных расходов, кроме хостинга. В то время как ноукод-платформы позволяют быстро запустить бота без знаний кода, но при этом ограничивают возможности настройки, зависят от работы сторонних сервисов и требуют регулярных платежей. Таким образом, выбор способа создания бота зависит от задач проекта и ресурсов команды.

1.4.2 Выбор языка программирования и базы данных

Выбор технологического стека и инструментов разработки является критическим этапом проектирования чат-бота, поскольку определяет возможности системы, сроки разработки и уровень поддержки.

Критерии выбора технологий для разработки чат-бота приемной кампании:

– Оперативность разработки. Необходимость в короткие сроки реализовать минимально жизнеспособный продукт (MVP), способный решать основные задачи.

– Масштабируемость и поддержка. Выбор решений, которые позволяют без существенных затрат времени добавлять новые функции и устранять недостатки.

– Интеграционная совместимость. Обеспечение технической возможности взаимодействия с внешними системами университета и популярными мессенджерами.

– Минимизация зависимости от ИИ-модулей. Применение проверенных, устойчивых технологий с предсказуемым поведением.

– Соответствие требованиям безопасности. Учет требований законодательства РФ в части обработки и хранения персональных данных.

– Экономическая эффективность. Оценка затрат на реализацию, поддержку и масштабирование решения.

Таблица 4 – Технологии проектирования чат-бота

Компонент	**Выбранные технологии**	**Обоснование выбора**
Язык программирования	Python	Широкое применение в разработке бэкенда, удобство работы с API и интеграций.
Фреймворк Backend	Flask	Легковесный и простой в использовании веб-фреймворк для быстрого создания REST API и маршрутов
Веб-интерфейс	HTML + CSS + JavaScript	Простая и гибкая реализация интерфейса чат-бота, включая поддержку голосового ввода и озвучивания.
Хранение FAQ-данных	Python dict (faq_data.py)	Быстрый и удобный способ хранения вопрос-ответ без необходимости базы данных.
Голосовой ввод и озвучивание	Web Speech API (SpeechRecognition и SpeechSynthesis	Встроенные браузерные API для распознавания речи и озвучивания текста, обеспечивают интерактивность.
Синтез речи (TTS)	pyttsx3	Локальная библиотека для озвучивания текста без необходимости подключения к интернет-сервисам
Обработка запросов пользователя	JavaScript (с динамическим отображением и отправкой запросов)	Удобство взаимодействия пользователя с чат-ботом, мгновенный отклик без перезагрузки страницы.
Визуальные элементы	CSS с фирменными цветами	Создание приятного и узнаваемого интерфейса с использованием корпоративных цветов.
Веб-страница	Flask + Jinja2 шаблоны	Удобный способ генерации HTML с возможностью динамического содержимого
Модальное окно FAQ	HTML + CSS + JavaScript	Удобная форма для отображения часто задаваемых вопросов без перехода на другую страницу.
Интеграция с мессенджерами	Telegram Bot API (возможная интеграция)	Поддержка популярных платформ для расширения доступности бота.
Обработка ошибок	traceback	Позволяет удобно отлавливать и логировать ошибки для быстрого поиска и исправления

Переходя к выбору технологии проектирования чат бота, рассмотрим языки программирования и базы данных, которые подходят для создания Telegram-бота. Существует несколько популярных вариантов, таких как Python, JavaScript (Node.js), Java, Go и другие.

– Python – язык программирования с простым и понятным синтаксисом, что облегчает обучение и ускоряет разработку. Он широко применяется в веб-разработке, науке о данных, машинном обучении и при создании чат-ботов. Благодаря большому количеству готовых библиотек (например, aiogram для Telegram-ботов), Python позволяет быстро реализовывать функционал [21].

– JavaScript (Node.js) – популярный язык для разработки веб-приложений, позволяет писать код как для клиентской, так и для серверной части. Node.js расширяет возможности JavaScript, делая его подходящим для создания серверных приложений, включая чат-ботов. Особенно удобен, если разработчик знаком с веб-технологиями [36].

– Язык программирования Java занимает устойчивые позиции в корпоративной разработке за счёт своей надёжности, масштабируемости и богатой экосистемы. Он широко используется для создания распределённых приложений и серверных решений. Хотя его синтаксис считается более сложным по сравнению с Python, именно эти особенности делают Java предпочтительным выбором для реализации сложных и устойчивых систем [37].

– Язык программирования Go, разработанный компанией Google, ориентирован на создание высокопроизводительных и надёжных сетевых приложений. Благодаря лаконичному синтаксису, высокой скорости выполнения и встроенным средствам параллелизма, Go становится всё более популярным в разработке ботов и микросервисной архитектуры [20].

Анализ современных языков программирования, применяемых для разработки чат-ботов, показал, что Python является оптимальным выбором, благодаря сочетанию простоты изучения, высокой скорости разработки и наличию обширного набора библиотек и инструментов, необходимых для

создания функциональных и надёжных решений. Большое сообщество разработчиков и активное развитие специализированных фреймворков для работы с обработкой естественного языка и интеграцией с мессенджерами обеспечивают удобство и эффективность процесса разработки.

Хотя другие языки, такие как JavaScript (Node.js), Java и Go, обладают определёнными преимуществами, они либо требуют более глубоких знаний и времени на освоение, либо уступают по количеству готовых решений, ориентированных на разработку чат-бота. Например, Java часто применяется в корпоративных средах и крупных проектах благодаря надёжности и масштабируемости, но её синтаксис и архитектура могут замедлять скорость прототипирования. Go, разработанный Google, обеспечивает высокую производительность и удобство в создании сетевых приложений, однако не обладает такой же широкой экосистемой для NLP-задач, как Python. JavaScript широко используется для веб-разработки и имеет гибкие возможности, но в сравнении с Python для некоторых задач, связанных с обработкой текста и интеграцией с ботами, может требовать дополнительных усилий.

Для разработки проекта выбран современный язык программирования Python, поскольку он обеспечивает необходимую гибкость для реализации всех функциональных требований и гарантирует стабильность работы чат-бота без зависимости от сторонних сервисов или дополнительных подписок. Использование языка программирования напрямую позволяет создавать уникальные и адаптированные под конкретные задачи решения, которые легко масштабировать и поддерживать. Это особенно важно для разрабатываемого проекта, где требуется оперативное обновление базы знаний и интеграция с различными коммуникационными платформами.

Таким образом, выбор Python обусловлен его универсальностью, широкими возможностями и удобством, как для разработки, так и для дальнейшего сопровождения программного продукта. Использование Python и его инструментов даёт гибкость в создании бота, возможность добавлять новые

функции и поддерживать проект без сложных настроек. [25]. Далее рассмотрим дополнительные технологии для разработки чат-бота:

– Flask – лёгкий и гибкий веб-фреймворк для Python, позволяющий быстро создавать веб-приложения и API с минимальными затратами ресурсов, идеально подходит для небольших и средних проектов [16].

– pyttsx3 – библиотека для синтеза речи, которая работает офлайн и позволяет озвучивать ответы без зависимости от внешних сервисов, обеспечивая быстрый отклик и стабильную работу [28].

– Обработка текста и поиск по базе FAQ (часто задаваемые вопросы) реализованы с простыми и эффективными алгоритмами, что снижает вероятность ошибок при интерпретации запросов и уменьшает нагрузку на поддержку.

– Многопоточность (threading) используется для озвучивания ответов в фоновом режиме, что улучшает отзывчивость приложения и пользовательский опыт.

Выделим технические и архитектурные решения для разрабатываемого чат-бота:

– Отказ от AI-модулей:

В статье «Введение в обработку естественного языка» подчеркивается, что «исключение сложных NLP-модулей обусловлено необходимостью обеспечить стабильность и простоту сопровождения» [8]. Обработка запросов строится на ключевых словах и шаблонах, что позволяет быстро настраивать и адаптировать логику бота.

– Модульность архитектуры:

Весь функционал разделён на независимые компоненты – обработчик диалогов, интеграционные службы, база знаний, веб-интерфейс, что позволяет безболезненно расширять и модифицировать систему.

– Безопасность:

Все коммуникации реализованы с применением защищённого протокола HTTPS, что предотвращает перехват данных. Дополнительно реализована проверка входящих данных (валидация) для снижения риска уязвимостей, таких как SQL-инъекции или XSS. Система логирования позволяет отслеживать действия пользователей и администраторов, обеспечивая прозрачность доступа и возможность анализа при инцидентах.

– Администрирование и обновления:

Редактирование сценариев, внесение изменений в базу знаний, а также управление контактной информацией и расписаниями осуществляется без необходимости полной перекомпиляции проекта или перезапуска сервера. Это достигается за счёт использования внешних файлов (например, JSON) или баз данных, что даёт гибкость и сокращает время обновлений.

Таким образом, простота архитектурного решения способствует оперативному развертыванию и последующим обновлениям приложения, не требуя сложных конфигураций. Это особенно актуально для небольших команд разработчиков и в условиях ограниченного времени на реализацию проекта.

2 Проектирование и реализация программного обеспечения чат-бота для информационного сопровождения приемной кампании ЛПИ – филиала СФУ

2.1 Архитектура программного обеспечения и логика работы чат-бота

Разработка программного решения требует предварительного определения структуры и логики взаимодействия компонентов. В данном проекте архитектура чат-бота строится на основе клиент-серверной модели с минимальной зависимостью от внешних платформ, что обеспечивает гибкость и расширяемость системы.

В основу архитектуры легли следующие компоненты:

– пользовательские интерфейсы представлены двумя способами взаимодействия: через Telegram-бота с использованием Bot API и через веб-приложение с клиентской частью на HTML и JavaScript. Это даёт возможность пользователям обращаться к боту как из мессенджера, так и с веб-страницы.

– веб-сервер, реализованный на Flask, выступает в роли центрального обработчика запросов. Он принимает сообщения, поступающие как от Telegram (через вебхуки), так и от веб-интерфейса (через HTTP-запросы), далее передаёт их в модуль обработки и возвращает сформированные ответы. Такой подход упрощает управление бизнес-логикой и обеспечивает единообразие работы с разными каналами.

– модуль логики обработки сообщений построен на функциях, которые сопоставляют входящий текст с базой заранее определённых вопросов и ответов, сохранённых в словаре. Этот способ позволяет легко расширять и адаптировать систему под новые запросы. Модуль обрабатывает как сообщения из Telegram, так и веб-запросы.

– модуль синтеза речи (TTS) преобразует текстовый ответ в аудиоформат. Для этого используется библиотека pyttsx3, работающая локально, без

подключения к внешним сервисам. Это уменьшает задержки и повышает автономность системы.

– данные и знания бота на текущем этапе хранятся в коде в виде словаря. Такой подход удобен для прототипа и небольшого объёма информации. В дальнейшем планируется использовать полноценную базу данных для хранения и обработки больших объёмов.

– конфигурация системы, включая токены и ключи доступа, размещена в отдельных файлах для удобства и безопасности.

В итоге проект имеет логическую структуру, которая обеспечивает простоту разработки и позволяет масштабировать систему в будущем. Все ключевые функции разнесены по независимым модулям, что облегчает сопровождение и развитие системы.

Рассмотрим взаимодействие компонентов Telegram. Архитектура проекта построена по принципу модульности и разделения ответственности между компонентами. Это обеспечивает стабильную работу, удобство сопровождения и возможность масштабирования в будущем. Ниже описана логика взаимодействия между основными элементами системы:

– пользователь взаимодействует с Telegram-ботом, отправляя текстовое сообщение в чат. Сообщение поступает через Telegram Bot API, который перенаправляет его на зарегистрированный вебхук.

– flask-сервер принимает входящий HTTP-запрос от Telegram и извлекает из него данные – текст сообщения. Этот компонент работает как связующее звено между внешним мессенджером и внутренней логикой проекта.

– блок логики обработки анализирует входящий текст. В текущей реализации это словарь с ключами в виде вопросов и соответствующими им ответами. Если сообщение совпадает с одним из шаблонов, возвращается готовый ответ. При необходимости можно внедрить алгоритмы машинного обучения или NLP-модуль для более сложной обработки.

– flask-сервер формирует итоговый ответ: отправляет пользователю текст и прикреплённый голосовой файл. Всё это возвращается в Telegram через Bot API.

Таким образом, весь цикл выглядит следующим образом:

Пользователь → telegram сообщение → flask-сервер → обработка → Пользователь. Данный цикл представлен на рисунке 10.

Рисунок 10 – Цикл работы компонентов в Telegram

Взаимодействие компонентов WEB:

– пользователь вводит вопрос на веб-странице и отправляет его через форму;

– веб-браузер формирует HTTP-запрос и отправляет данные на Flask-сервер. Этот компонент служит посредником между пользователем и внутренней логикой приложения;

– блок логики обработки анализирует полученный текст. В текущей реализации используется словарь с вопросами и ответами. При совпадении с шаблоном возвращается соответствующий ответ. При необходимости можно подключить более сложные алгоритмы обработки естественного языка;

– модуль синтеза речи (TTS) преобразует текстовый ответ в аудиофайл с помощью библиотеки pyttsx3, работающей локально, что уменьшает задержки и обеспечивает автономность;

– flask-сервер формирует окончательный ответ и возвращает его в виде JSON с текстом и ссылкой на голосовой файл. Браузер отображает ответ пользователю и воспроизводит аудио.

Цикл будет выглядеть следующим образом:

Пользователь → Flask-сервер → FAQ-словарь → ответ + синтез речи → ответ (json) → пользователь. Данный цикл показан на рисунке 11.

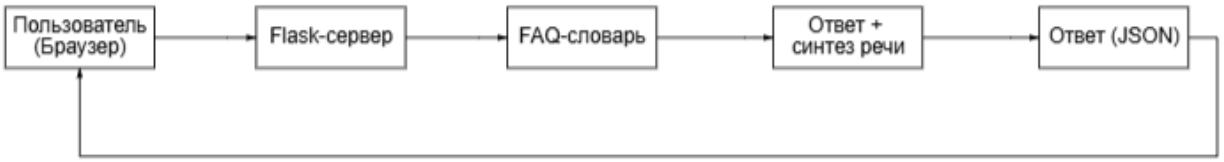

Рисунок 11 – Цикл работы компонентов в WEB

Исходя из архитектуры проекта и описанной логики взаимодействия компонентов, можно выделить ключевые функциональные модули, которые обеспечивают работу чат-бота и веб-приложения. Каждый из них выполняет отдельную задачу, что позволяет поддерживать модульность и упрощает развитие и масштабирование системы.

Рассмотрим основные модули проектирования разрабатываемого чат-бота:

– *модуль приёма сообщений* отвечает за получение запросов от пользователей. Для Telegram это обработка входящих сообщений через вебхук. Для веб-версии – приём POST-запросов с текстом пользователя на сервер Flask:

– *модуль обработки текста* выполняет анализ входящего сообщения. В проекте реализован через поиск ключевых слов в FAQ-словаре, что позволяет сопоставить вопрос с предопределённым ответом. В дальнейшем этот модуль можно расширить, подключив NLP или машинное обучение:

– *модуль генерации ответа* формирует финальный ответ на основе результата обработки. Может возвращать только текст, либо текст с дополнительной информацией (например, ссылкой):

– *модуль синтеза речи (TTS)* преобразует текстовый ответ в аудиоформат. В проекте используется локальная библиотека pyttsx3, что обеспечивает автономность и отсутствие задержек, связанных с вызовом внешних API. В Telegram этот модуль отключён, а в веб-версии используется для создания голосового сопровождения ответа:

– *модуль отправки ответа* отвечает за передачу сформированного ответа пользователю. В Telegram это отправка сообщений через Bot API. В веб-приложении – возврат JSON-ответа с текстом и ссылкой на аудиофайл.

Таким образом, выделенные модули обеспечивают чёткое разделение функциональности и упрощают сопровождение проекта. Такая модульная структура позволяет легко адаптировать систему под разные каналы взаимодействия с пользователем – будь то Telegram или веб-интерфейс.

Для наглядного представления последовательности обработки запросов и взаимодействия компонентов системы на рисунке 12 представлена блок-схема, демонстрирующая основные этапы работы чат-бота. Эта схема иллюстрирует логику приема входящих сообщений, обработки запросов, генерации ответов и, при необходимости, озвучивания текста, что позволяет лучше понять структуру и функционирование всей системы в целом

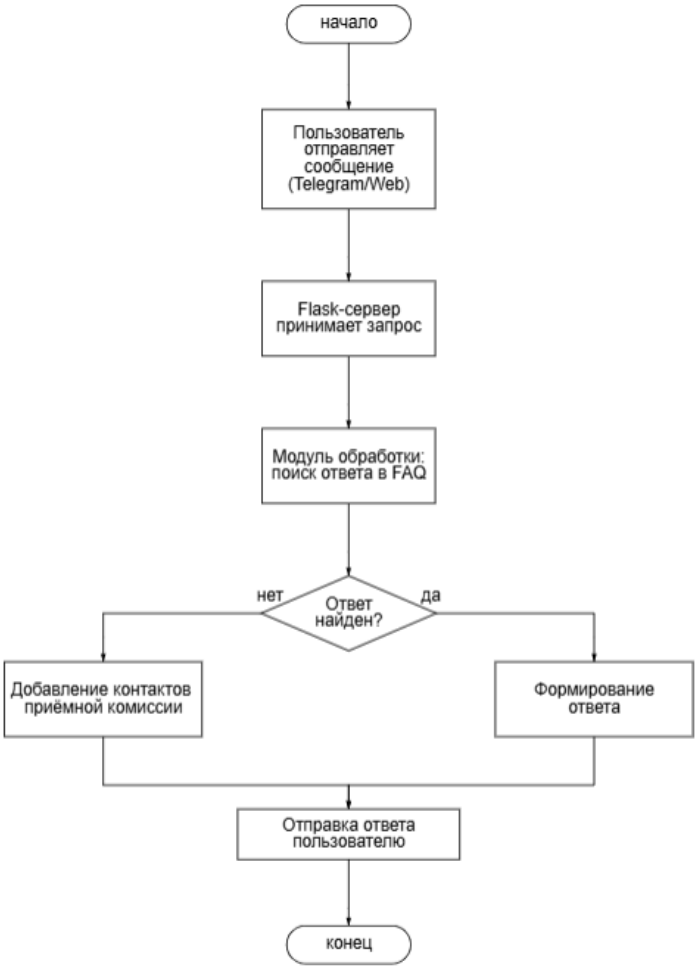

Рисунок 12 – Блок-схема работы чат-бота

Данная блок-схема способствует более глубокому пониманию логики обработки запросов и наглядно показывает, как система реагирует на различные сценарии, включая случаи отсутствия готового ответа и передачу контактов приёмной кампании для дальнейшего взаимодействия.

2.2 Реализация чат-бота для информационного сопровождения приемной кампании ЛПИ – филиала СФУ

Реализация чат-бота приёмной кампании выполнена с использованием веб-фреймворка Flask, который обеспечивает обработку входящих HTTP-запросов и взаимодействие с пользователем через веб-интерфейс. Для озвучивания ответов применяется библиотека pyttsx3, позволяющая синтезировать речь непосредственно на стороне сервера, что обеспечивает автономность работы и минимизирует задержки при воспроизведении голосовых сообщений.

Код импорта библиотек и модулей представлен на рисунке 13.

```
from flask import Flask, render_template, request, jsonify
import pyttsx3
from faq_data import faq
import traceback
import threading
```

Рисунок 13 – Код импорта библиотек и модулей

В начале файла происходит импорт необходимых библиотек и модулей, обеспечивающих работу приложения. Для обработки веб-запросов используется Flask – лёгкий и гибкий веб-фреймворк на Python, который позволяет быстро создавать веб-приложения и RESTful API. В данном проекте Flask отвечает за приём и обработку HTTP-запросов, а также за формирование ответов пользователям.

Для озвучивания текстовых ответов применяется библиотека pyttsx3, которая обеспечивает синтез речи непосредственно на стороне сервера без необходимости подключения к интернету. Это повышает автономность работы и снижает задержки при воспроизведении голосовых сообщений.

Данные с часто задаваемыми вопросами и ответами вынесены в отдельный модуль faq_data.py и загружаются в переменную faq. Структура данных представлена в виде словаря, где ключами выступают вопросы или ключевые слова, а значениями – текстовые ответы либо словари с дополнительной информацией, например, ссылками.

Модуль traceback используется для удобного логирования и отслеживания ошибок. В случае возникновения исключений он позволяет определить место и причину сбоя, что облегчает процесс отладки и поддержки.

Для реализации озвучивания без блокировки основного потока обработки запросов применяется стандартный модуль threading. Он позволяет запускать процесс синтеза речи в отдельном потоке, обеспечивая плавную и эффективную работу сервера.

После подключения всех необходимых библиотек и модулей переходим к следующему этапу – созданию и настройке самого Flask-приложения. Этот шаг является ключевым для обеспечения корректной маршрутизации входящих запросов и интеграции всех компонентов системы, включая модуль синтеза речи и обработку данных из базы часто задаваемых вопросов.

Код создание и настройка Flask-приложения представлен на рисунке 14.

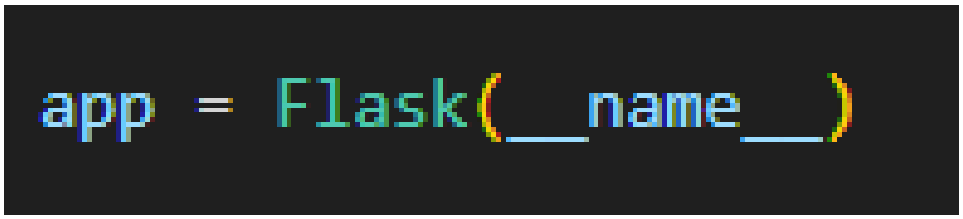

Рисунок 14 – Код создание и настройка Flask-приложения

На этом этапе создаётся экземпляр Flask-приложения, который будет служить основным объектом для обработки всех входящих HTTP-запросов.

Переменная app является центральным элементом веб-приложения и отвечает за регистрацию маршрутов (routes), настройку обработки запросов, а также управление жизненным циклом приложения. Через этот объект реализуется вся логика взаимодействия с клиентами, включая получение данных, их обработку и формирование ответов.

После создания экземпляра Flask-приложения следующей важной задачей становится организация синтеза речи, который позволит чат-боту озвучивать свои ответы. Для этого необходимо корректно инициализировать движок TTS и обеспечить безопасную многопоточную работу.

Код инициализации движка озвучивания (TTS) и создание блокировки представлен на рисунке 15.

```
# Инициализация TTS и блокировка для синхронизации
tts_engine = pyttsx3.init()
tts_lock = threading.Lock()
```

Рисунок 15 – Код инициализации движка озвучивания (TTS) и создание блокировки

Для реализации функции озвучивания текста необходимо создать экземпляр движка синтеза речи (TTS). В pyttsx3 это делается вызовом init(). После инициализации у нас есть объект tts_engine, который умеет преобразовывать текст в речь.

Так как pyttsx3 не является потокобезопасным (нельзя параллельно несколько раз запускать озвучку в разных потоках без риска конфликтов), мы создаём объект блокировки tts_lock. Эта блокировка гарантирует, что в каждый момент времени озвучивание выполняется только одним потоком, предотвращая возможные сбои и искажения звука.

Когда готова настройка озвучивания, следующим этапом становится реализация механизма поиска ответов в базе FAQ. Это сердце логики чат-бота – от правильности его работы зависит качество взаимодействия с пользователем.

Код функции поиска ответа с учётом длины совпадения ключа представлен на рисунке 16.

```python
# Функция поиска с приоритетом по длине совпадения
def get_answer_with_link(question):
    question = question.lower().strip()
    matched = []
    for key, answer in faq.items():
        if key.lower() in question:
            matched.append((len(key), answer))
    if matched:
        matched.sort(key=lambda x: x[0], reverse=True)
        answer = matched[0][1]
        if isinstance(answer, dict):
            # Возвращаем только 'text' и 'link' если есть
            text = answer.get('text', '')
            link = answer.get('link', '')
            # Если нет отдельного link, возвращаем пустую строку
            return text, link
        else:
            return answer, ''
    return "Извините, я пока не знаю ответа на этот вопрос.", ''
```

Рисунок 16 – Код функции поиска ответа с учётом длины совпадения ключа

Эта функция является ядром логики поиска ответа на заданный пользователем вопрос. Она принимает строку question и пытается найти в словаре faq все ключи, которые встречаются внутри этого вопроса. Для этого мы приводим и вопрос, и ключи к нижнему регистру – так поиск становится нечувствительным к регистру.

Чтобы повысить точность, мы учитываем длину совпавшего ключа – чем длиннее ключ, тем более точное совпадение. Все совпадения собираются в список matched с длиной ключа и самим ответом.

После этого список сортируется по длине ключа по убыванию, и мы берём самый длинный (самый релевантный) ответ.

Интересный момент – в словаре faq ответы могут быть либо строками, либо словарями, содержащими поля text и link. Если это словарь, то мы возвращаем оба поля отдельно, чтобы потом можно было выводить текст и, при необходимости, ссылку. Если нет, возвращаем просто строку и пустую ссылку.

Если же совпадений не найдено, функция возвращает стандартный ответ – «Извините, я пока не знаю ответа на этот вопрос.».

Получив механизм поиска ответов, необходимо реализовать функцию, которая будет воспроизводить эти ответы голосом, строго по одному потоку. Это позволит избежать конфликтов и обеспечит стабильную звуковую отдачу.

Код функции озвучивания текста представлен на рисунке 17.

```
# Функция для озвучивания
def speak_answer(answer):
    with tts_lock:
        tts_engine.say(answer)
        tts_engine.runAndWait()
```

Рисунок 17 – Код функции озвучивания текста с использованием блокировки

Эта функция отвечает за воспроизведение звукового ответа. Важным моментом здесь является использование блокировки tts_lock. Мы помещаем вызовы озвучивания в контекст менеджера with, чтобы гарантировать, что в любой момент времени только один поток может управлять движком озвучивания.

Внутри блокировки сначала передаём текст в движок методом say, а затем вызываем runAndWait(), который выполняет синтез речи и

воспроизведение. Без этой блокировки возможны ошибки или наложение звуков, если несколько запросов пытаются озвучить ответы одновременно.

После настройки озвучивания и логики поиска можно переходить к созданию маршрутов для обработки пользовательских запросов. Первый из них – это маршрут для главной страницы, которая служит точкой входа во взаимодействие.

Код с маршрутом для главной страницы веб-приложения представлен на рисунке 18.

```
@app.route('/')
def index():
    return render_template('index.html')
```

Рисунок 18 – Код маршрута для главной страницы веб-приложения

Здесь мы описываем обработчик GET-запроса по адресу / (корень сайта). Когда пользователь открывает страницу, сервер отдает ему HTML-шаблон index.html.

Этот файл должен содержать клиентскую часть сайта – интерфейс, куда пользователь вводит вопрос и видит ответ. Flask автоматически найдёт и отдаст этот файл из папки templates.

Когда клиент получает интерфейс, следующим шагом становится настройка обработки POST-запросов, в которых пользователь отправляет свои вопросы. Именно здесь начинается логика взаимодействия между фронтендом и сервером.

Код для обработки POST-запроса с вопросом пользователя представлен на рисунке 19.

Рисунок 19 – Код обработки POST-запроса с вопросом пользователя

В этом маршруте происходит обработка POST-запроса на адрес /ask, который приходит от клиентской части с JSON-данными, содержащими вопрос пользователя и режим ответа (text или link).

Сначала из полученного JSON извлекается вопрос и режим ответа. Если вопрос пустой, сервер возвращает ошибку с просьбой задать вопрос.

Далее вызывается функция get_answer_with_link, которая ищет наиболее подходящий ответ на вопрос.

Если выбран режим "text", то ответ отправляется обратно клиенту в формате JSON, а также асинхронно запускается озвучивание этого ответа в отдельном потоке, чтобы не блокировать обработку других запросов.

Если выбран режим "link", сервер возвращает текст ответа и соответствующую ссылку, либо подставляет ссылку на поиск в Яндексе, если прямой ссылки нет.

В случае ошибки или неизвестного режима возвращается соответствующее сообщение с кодом ошибки.

Когда все маршруты определены, остаётся завершить серверную часть запуском самого Flask-приложения, включив отладку и многопоточность для обеспечения устойчивости под нагрузкой.

Код для запуска приложения представлен на рисунке 20.

```python
if __name__ == '__main__':
    app.run(debug=True, threaded=True)
```

Рисунок 20 – Код запуска приложения

В конце файла происходит запуск Flask-приложения с включенным режимом отладки и поддержкой многопоточности. Это позволяет обрабатывать несколько запросов одновременно, что улучшает отзывчивость и масштабируемость чат-бота.

Переходя от серверной логики к клиентской части, рассмотрим HTML-документ, который определяет внешний вид интерфейса чат-бота и обеспечивает взаимодействие пользователя с системой.

Основа кода HTML представлена на рисунке 21.

```html
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Чат-бот lpibot</title>
    <style>
        /* Стилизация страницы */
    </style>
</head>
<body>
    <!-- Основное содержимое страницы -->
</body>
</html>
```

Рисунок 21 – Основа кода HTML

<meta charset="UTF-8"> – этот тег задаёт кодировку страницы в формате UTF-8, который является стандартом для современных веб-страниц и поддерживает отображение большинства языков, включая русский, обеспечивая корректное отображение символов.

<title>Чат-бот lpibot</title> – определяет заголовок страницы, который отображается во вкладке браузера и используется поисковыми системами для индексации, помогая пользователю понять содержание страницы. <style> ... </style> – блок встроенных CSS-стилей, в котором прописываются правила оформления элементов страницы: цвета, шрифты, отступы, размеры кнопок и других интерфейсных компонентов. Это позволяет создать уникальный и удобный дизайн, улучшая восприятие сайта пользователем.

Основное содержимое веб-страницы располагается внутри тега <body>, где формируется весь пользовательский интерфейс чат-бота.

Часть клиентского интерфейса представляет собой модальное окно часто задаваемых вопросов (FAQ), предоставляющее быстрый доступ к справочной информации без необходимости писать запрос в чат.

– Модальное окно FAQ представляет собой всплывающее окно, которое появляется поверх основного контента страницы. Его назначение – предоставить пользователю быстрый доступ к списку часто задаваемых вопросов и ответов на них (FAQ – Frequently Asked Questions), облегчая получение информации без необходимости покидать текущую страницу. Это позволяет пользователю быстро получить нужную информацию без необходимости писать вопрос в чат. Само окно представлено на рисунке 22.

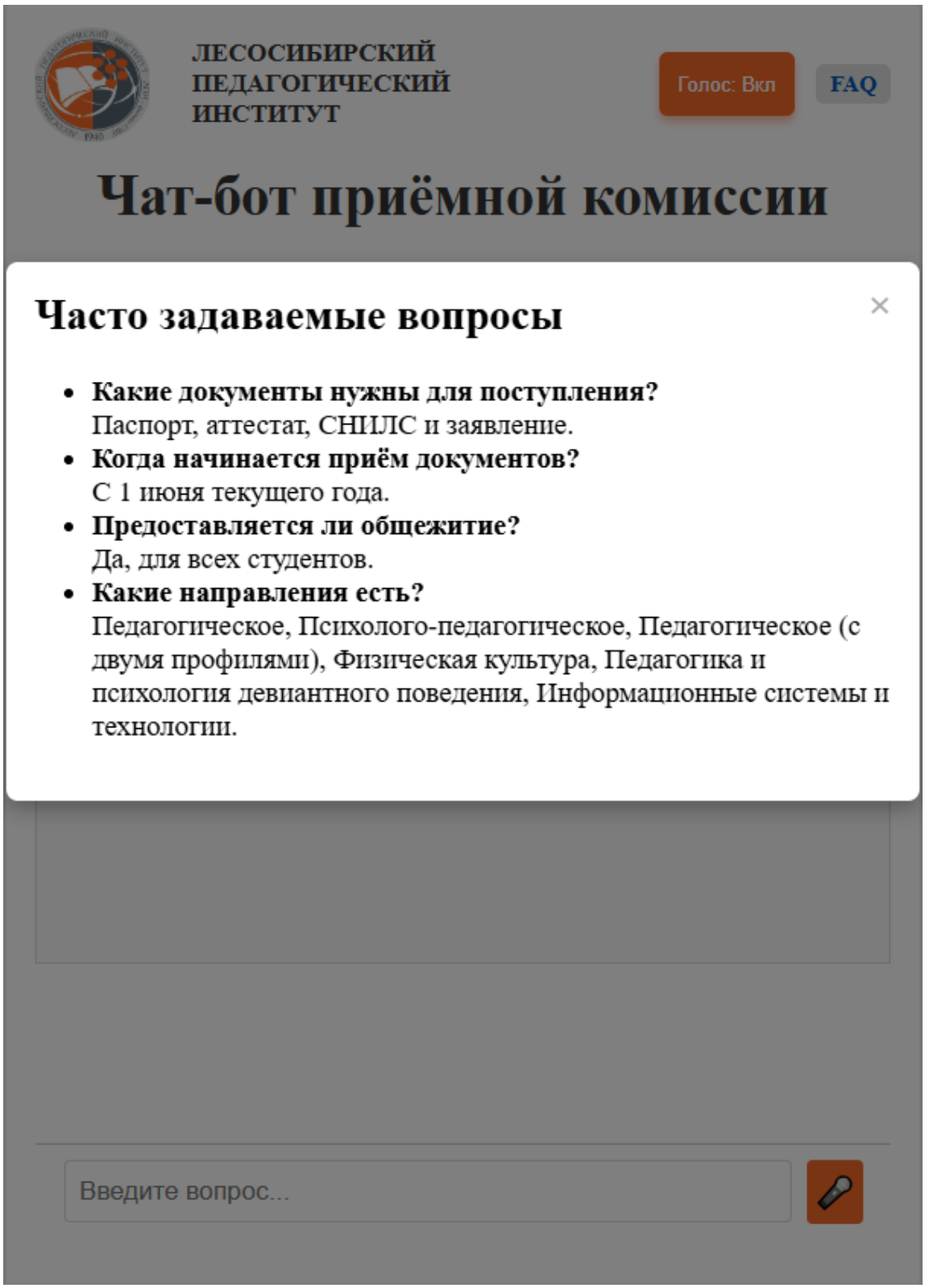

Рисунок 22 – Часто задаваемые вопросы (FAQ)

HTML-код кнопки FAQ представлен на рисунке 23:

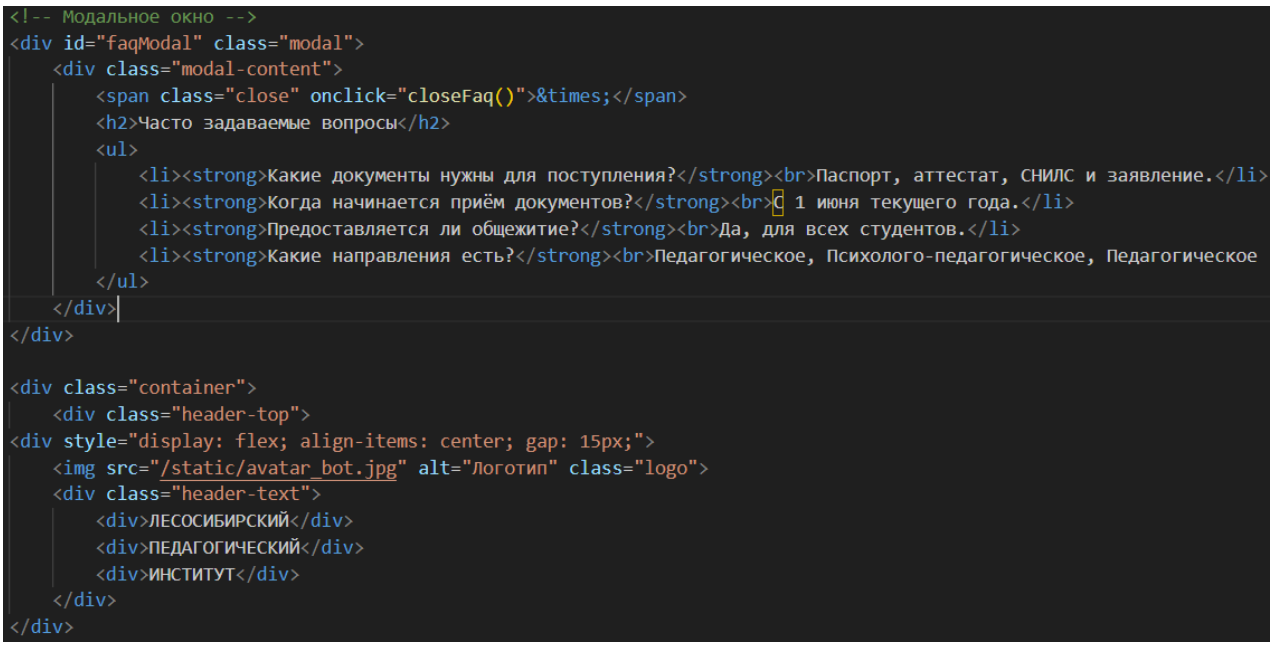

Рисунок 23 – Код часто задаваемые вопросы (FAQ)

–

Весь модальный блок обёрнут в контейнер с ID faqModal и классом modal. Это позволяет управлять его отображением через CSS и JavaScript – например, скрывать или показывать при необходимости.

–

Внутренний блок, содержащий всё содержимое окна: заголовок, список вопросов и кнопку закрытия. Он стилизуется отдельно (например, с рамками, фоном, тенями).

– Кнопка закрытия (×)

Символ «×» выполняет роль кнопки «Закрыть». При клике вызывается функция closeFaq(), которая, скорее всего, скрывает модальное окно.

– – Список FAQ

Содержит перечень часто задаваемых вопросов и кратких, информативных ответов. Каждый вопрос выделен жирным шрифтом (), а под ним – ответ.

Это окно помогает пользователю быстро получить ответы на типовые вопросы без необходимости обращаться к сотрудникам или искать

информацию на сайте. Примеры вопросов – про документы, сроки приёма, наличие общежития, направления подготовки.

В дополнение к модальному окну, на странице размещена кнопка управления озвучиванием, позволяющая пользователю включать или отключать голосовой ответ по своему усмотрению.

Код для кнопки включения/отключения озвучки представлен на рисунке 24.

```
<button id="voiceToggleBtn" class="voice-toggle-btn" title="Вкл/Выкл озвучивание">Голос: Вкл</button>
```

Рисунок 24 – Код кнопки включения/отключения озвучки

Эта кнопка предназначена для управления функцией озвучивания ответов чат-бота. Она позволяет пользователю включать или отключать голосовой синтез, в зависимости от своих предпочтений – например, если пользователь хочет слушать ответы, а не только читать их на экране.

id="voiceToggleBtn" – уникальный идентификатор кнопки, используется в JavaScript для отслеживания и изменения её состояния при взаимодействии.

class="voice-toggle-btn" – CSS-класс, который отвечает за визуальное оформление кнопки (цвет, размер, шрифт и пр.).

title="Вкл/Выкл озвучивание" – всплывающая подсказка, появляющаяся при наведении курсора. Она сообщает, что кнопка управляет озвучиванием.

Поведение при нажатии:

При клике по кнопке:

Срабатывает JavaScript-функция, которая:

Меняет внутреннюю переменную или флаг (например, isVoiceOn) на противоположное значение.

Обновляет текст кнопки на «Голос: Выкл», если озвучивание выключается, или обратно на «Голос: Вкл», если включается.

Эти действия дают пользователю обратную связь и одновременно обновляют функциональность: дальнейшие ответы либо озвучиваются, либо нет.

Такой подход делает интерфейс более доступным и удобным, особенно для пользователей с ограниченным зрением или тех, кто предпочитает голосовой вывод. Данная кнопка представлена на рисунке 25.

Рисунок 25 – Кнопка включения/отключения озвучки

Основой взаимодействия с чат-ботом служит поле ввода вопроса и кнопка активации микрофона. Это позволяет пользователю выбрать удобный способ ввода – текстом или голосом.

Код поля ввода вопроса и кнопка микрофона представлен на рисунке 26.

```
<div class="input-row">
    <input type="text" id="question" placeholder="Введите вопрос..." oninput="showModeButtons()">
    <button id="speechButton" onclick="startSpeechRecognition()" title="Это микрофон">🎤</button>
</div>
```

Рисунок 26 – Код поля ввода вопроса и кнопка микрофона

Основное поле ввода (<input>):

– Это текстовое поле, в которое пользователь вводит свой вопрос или запрос к чат-боту.

– Атрибут id="question" даёт уникальный идентификатор, который используется в JavaScript для получения и изменения содержимого этого поля.

– placeholder="Введите вопрос..." – это подсказка, которая отображается в поле до того, как пользователь начнёт вводить текст. Она помогает понять, что именно здесь нужно написать.

– Атрибут oninput="showModeButtons()" – при каждом изменении содержимого поля (например, при вводе, удалении или вставке текста) вызывается функция showModeButtons(). Эта функция отвечает за отображение дополнительных элементов интерфейса – кнопок выбора режима ответа (текст или ссылка). Таким образом, если поле пустое, кнопки не видны, а при вводе текста они появляются, улучшая интерактивность и удобство.

Кнопка микрофона (<button>):

– Эта кнопка служит для активации голосового ввода – то есть пользователь может вместо того, чтобы печатать, просто произнести вопрос вслух.

– Атрибут id="speechButton" служит для обращения к кнопке в JavaScript.

– При клике на кнопку вызывается функция startSpeechRecognition(), которая запускает процесс распознавания речи.

– Атрибут title="Это микрофон" показывает подсказку при наведении курсора на кнопку – это помогает пользователям понять, что кнопка отвечает за голосовой ввод.

– Внутри кнопки размещён символ «🎤», визуально указывающий на микрофон.

Само поле ввода и кнопка микрофона расположено на рисунке 27.

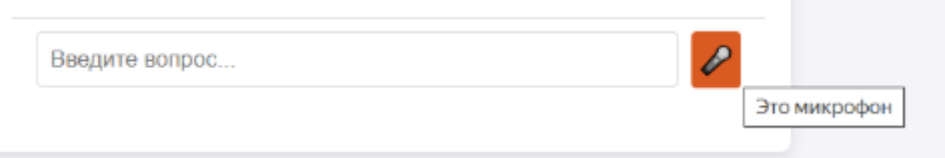

Рисунок 27 – Поле ввода и кнопка микрофон

Чтобы дополнительно повысить гибкость, интерфейс предлагает выбор между получением полного текста ответа и ссылки на источник. Этот выбор реализован через динамически появляющиеся кнопки.

Код кнопок для выбора режима ответа представлен на рисунке 28.

```
<div id="modeButtons" class="button-container">
    <button onclick="askQuestion('text')">Ответ получить в окне</button>
    <button onclick="askQuestion('link')">Получить ссылку</button>
</div>
```

Рисунок 28 – Код кнопки выбора режима ответа

Данный блок содержит две кнопки, предназначенные для выбора пользователем формата получения ответа от чат-бота. Этот элемент интерфейса повышает удобство взаимодействия, предоставляя гибкость в способе получения информации.

Первая кнопка – «Ответ получить в окне» – при нажатии выводит полный текстовый ответ непосредственно в окне чата. Такой формат подходит для быстрого ознакомления с информацией, когда пользователю важно получить развернутый ответ без дополнительных переходов.

Вторая кнопка – «Получить ссылку» – предлагает пользователю ссылку на внешний или внутренний ресурс, содержащий более подробную информацию по заданному вопросу. Это удобно, если ответ слишком объемный или требует просмотра дополнительных материалов, расположенных за пределами основного интерфейса чат-бота.

Рассмотрим поведение и динамику появления кнопок

Кнопки появляются динамически, только тогда, когда в поле ввода вопроса уже есть текст, данная механика представлена на рисунке 29. Это реализуется функцией showModeButtons(), которая отслеживает ввод пользователя в поле и управляет видимостью блока с кнопками.

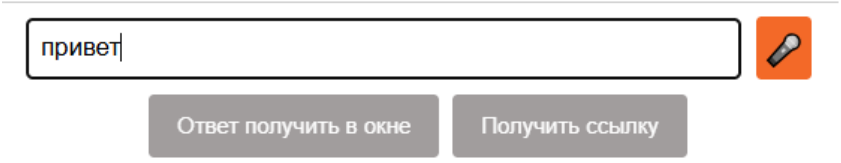

Рисунок 29 – Кнопки выбора режима ответа

Если поле пустое, кнопки скрыты, чтобы не отвлекать пользователя и не создавать излишнюю нагрузку на интерфейс, представлен на рисунке 30. Такой подход улучшает восприятие и делает интерфейс более чистым и удобным.

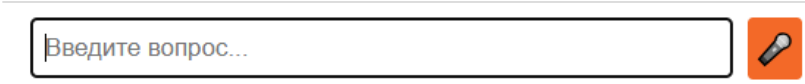

Рисунок 30 – Отсутствие кнопок при пустом поле

Наконец, завершает клиентскую часть встроенная в браузер система синтеза речи – SpeechSynthesis API, которая воспроизводит текст с заданными параметрами и обеспечивает голосовую отдачу прямо на стороне клиента.

Код для озвучивания текста представлен на рисунке 31.

```
function speakText(text) {
    if (!voiceEnabled) return;  // Если голос выключен — ничего не озвучиваем

    // остальной код озвучивания...
    if (!voices.length) {
        voices = window.speechSynthesis.getVoices();
    }

    const utterance = new SpeechSynthesisUtterance(text);
    // ...
    window.speechSynthesis.speak(utterance);
    }
    if (!voices.length) {
        voices = window.speechSynthesis.getVoices();
    }

    const utterance = new SpeechSynthesisUtterance(text);
    const selectedVoice = voices.find(voice =>
        voice.lang.startsWith('ru') &&
        (voice.name.toLowerCase().includes('google') ||
        voice.name.toLowerCase().includes('microsoft') ||
        voice.name.toLowerCase().includes('anna') ||
        voice.name.toLowerCase().includes('irina'))
    );

    if (selectedVoice) {
        utterance.voice = selectedVoice;
    }

    utterance.pitch = 1.2;
    utterance.rate = 0.9;
    window.speechSynthesis.speak(utterance);
```

Рисунок 31 – Код озвучивания текста (Speech Synthesis)

SpeechSynthesisUtterance – это интерфейс Web Speech API, представляющий речь, которая должна быть произнесена синтезатором речи. Он содержит текст, который необходимо озвучить, а также параметры озвучивания, такие как голос, скорость (rate), высота тона (pitch) и громкость. Этот объект передается в движок синтеза речи браузера для воспроизведения аудио [19].

Логика работы функции

– Проверка состояния озвучивания

В начале функции проверяется флаг voiceEnabled, который определяет, включено ли озвучивание. Если озвучка отключена (значение false), функция немедленно завершается и не запускает синтез речи, что экономит ресурсы и не мешает пользователю.

– Получение списка доступных голосов

Переменная voices содержит массив доступных голосов, предоставляемых браузером. Если этот список ещё не загружен (пустой), функция запрашивает его через window.speechSynthesis.getVoices(). Это важно, так как список голосов загружается асинхронно, и его нужно получить перед выбором конкретного голоса.

– Создание объекта для озвучивания

Для воспроизведения текста создаётся экземпляр SpeechSynthesisUtterance, которому передаётся сам текст. Этот объект отвечает за управление параметрами синтеза и собственно озвучивание.

– Выбор подходящего русского голоса

Функция пытается найти в списке голосов подходящий голос для русского языка. Для этого она фильтрует по языковому коду (voice.lang.startsWith('ru')) и отдаёт приоритет голосам с именами, содержащими популярные русскоязычные варианты: google, microsoft, anna, irina. Это позволяет обеспечить качественное и понятное произношение для русского текста.

– Настройка параметров голоса

Если подходящий голос найден, он устанавливается для объекта озвучивания.

Далее настраиваются параметры озвучивания:

pitch (высота голоса) выставляется в 1.2, что делает голос чуть более звонким и приятным.

rate (скорость речи) задаётся как 0.9, немного медленнее стандартной скорости, чтобы текст было легче воспринимать.

– Запуск озвучивания

Наконец, функция вызывает метод window.speechSynthesis.speak(), который запускает процесс воспроизведения речи с выбранными параметрами.

Разработанный чат-бот приёмной кампании представляет собой полноценное веб-приложение с поддержкой голосового взаимодействия, что делает его удобным и доступным для широкого круга пользователей. Использование фреймворка Flask позволило организовать гибкую и масштабируемую серверную часть, обеспечивающую быструю обработку запросов. Интеграция с библиотекой pyttsx3 дала возможность реализовать офлайн-озвучивание ответов, повышая автономность и надёжность системы.

Таким образом, реализованный чат-бот способен эффективно справляться с задачами предоставления справочной информации абитуриентам, снижая нагрузку на сотрудников приёмной кампании и повышая доступность консультаций в режиме 24/7.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы разработан чат-бот для приёмной кампании ЛПИ филиала СФУ, предназначенный для информационного сопровождения пользователей. Созданное программное решение автоматизирует процесс предоставления актуальной информации абитуриентам, студентам и сотрудникам, что способствует повышению качества и оперативности обслуживания.

Актуальность разработки обусловлена возрастающей потребностью в автоматизации бизнес-процессов и снижении нагрузки на операторов в условиях большого потока обращений, особенно в период приёмной кампании. Использование чат-бота позволяет минимизировать трудности, связанные с поиском необходимой информации, обеспечивая быстрый доступ к ответам на часто задаваемые вопросы.

В ходе исследования были изучены существующие решения и подходы к созданию чат-ботов, проанализированы потребности целевой аудитории и определены оптимальные программные средства для реализации проекта. Разработана архитектура и структура чат-бота, осуществлена его программная реализация с использованием языка Python. Проведено тестирование, подтвердившее работоспособность и удобство взаимодействия с системой.

Применённые методы – теоретический анализ, аналитический подход, программирование, экспериментальное моделирование, тестирование и сравнительный анализ – позволили комплексно подойти к решению поставленных задач и достичь целей исследования.

Результаты работы были представлены на нескольких научных конференциях, а также опубликованы в сборнике научных трудов, что свидетельствует о практической значимости и актуальности проведённого исследования.

Таким образом, созданный чат-бот является эффективным инструментом для автоматизации информационного сопровождения в приёмной кампании

ЛПИ филиала СФУ и может быть рекомендован к внедрению в образовательную организацию для улучшения качества взаимодействия с пользователями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Автоматизация бизнеса [Электронный ресурс]. URL: https://www.kp.ru/guide/avtomatizatsija-biznesa.html (дата обращения: 19.06.2025).

2. Аудитория Telegram: исследование Mediascope [Электронный ресурс]. – URL: https://mediascope.net/news/1601603/ (дата обращения: 03.01.2025).

3. Биккулова, О. С. Чат-бот в методике преподавания РКИ / О. С. Биккулова, М. И. Ивкина. // Мир русского слова. – 2021. – № 1. – С. 91–96 : ил. – Библиогр. в конце ст. – ISSN 1811-1629.

4. Chatfuel. Создание чат-ботов без кода // Chatfuel – URL: https://chatfuel.com/ (дата обращения: 18.06.2025).

5. BotFather. Справка Telegram // Telegram Bot Platform. – URL: https://core.telegram.org/bots#botfather (дата обращения: 18.06.2025).

6. BotTap. Платформа для создания Telegram-ботов // BotTap – URL: https://bottap.me/ (дата обращения: 18.06.2025).

7. Валиев Р. М. Что такое чат-боты / Р. М. Валиев, Г. О. Буглов, А. А. Сафонова, Е. В. Янковская. // ВАВТ, кафедра менеджмента и маркетинга; материалы исследования. – 2024. – 25 с.

8. Введение в обработку естественного языка / автор не указан // GeeksforGeeks. – 2023. – (дата обращения: 18.06.2025).

9. VK API. Документация // VK Developers. – URL: https://dev.vk.com/api (дата обращения: 17.06.2025).

10. Горячкин, Б. С. Эффективность использования чат-ботов в образовательном процессе / Б. С. Горячкин, Д. А. Галичий, В. С. Цапий, В. В. Бурашников, Т. Ю. Крутов. // Компьютерные и информационные науки. – 2023. – С. 1–23. – УДК 004.42.

11. Гринберг, М. Flask Web Development: Developing Web Applications with Python / М. Гринберг. – 2-е изд. – О’Райли Медиа, 2018. – 258 с. – ISBN 978-1491991732.

12. Гюльдал, Х., Динчер, Э. О. Можно ли считать чат-боты с правилами приемлемой альтернативой для студентов высших учебных заведений? / Х. Гюльдал, Э. О. Динчер. // Образование и информационные технологии. – 2025. – Т. 30, № 3. – С. 3979–4012.

13. Джанарсанам, С. Разработка чат-ботов и разговорных интерфейсов: чат-боты и голосовые пользовательские интерфейсы на платформах Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio и Alexa Skills / С. Джанарсанам; пер. с англ. М. А. Райтман. – Москва : ДМК Пресс, 2019. – 340 с. – ISBN 978-5-97060-542-4.

14. Донован, А. А., Керниган, Б. В. Язык программирования Go / А. А. Донован, Б. В. Керниган. – Санкт-Петербург : Питер, 2015. – 400 с. – ISBN 978-0-134-19044-0.

15. Елисеев, А. И., Минин, Ю. В., Гриднев, В. А. Разработка веб-приложений с использованием фреймворка Flask. Ч. 1 : учебное пособие / А. И. Елисеев, Ю. В. Минин, В. А. Гриднев. – Тамбов : ТГТУ, 2020. – 82 с. – ISBN 978-5-8265-2188-5.

16. Елисеев, А. И., Минин, Ю. В., Гриднев, В. А. Разработка веб-приложений с использованием фреймворка Flask. Ч. 2 : учебное пособие / А. И. Елисеев, Ю. В. Минин, В. А. Гриднев. – Тамбов : ТГТУ, 2021. – 84 с. – ISBN 978-5-8265-2438-1.

17. Заяц, А. М., Васильев, Н. П. Проектирование и разработка WEB-приложений. Введение в frontend и backend разработку на JavaScript и node.js : учебное пособие для СПО / А. М. Заяц, Н. П. Васильев. – 3-е изд., стер. – Санкт-Петербург : Лань, 2023. – 120 с. – ISBN 978-5-507-45423-5.

18. Источник: MDN Web Docs, SpeechSynthesisUtterance (дата обращения: 18.06.2025).

19. Киселев, А. Н. Программирование на Go. Разработка приложений XXI века / А. Н. Киселев, М. Саммерфильд ; пер. с англ. А. Н. Киселева. – 2-е изд., электронное. – Москва : ДМК Пресс, 2023. – 581 с. – ISBN 978-5-89818-611-1.

20. Крусс, И. А. Развитие технологий искусственного интеллекта в банковском секторе / И. А. Крусс // Банковское дело. – 2022. – Т. 9. – С. 62–65. – ISSN 2071-4904.

21. Лутц, М. Изучаем Python / М. Лутц. – 5-е изд. – Москва : Вильямс, 2013. – 1540 с. – ISBN 978-1-449-35573-9.

22. Лутц, М. Изучаем Python / М. Лутц. – Москва: Вильямс, 2022. – 702 с. – ISBN 978-5-8459-3149-9.

23. Manybot. Официальный сайт // Manybot – Telegram Bot Builder. – URL: https://manybot.io/ (дата обращения: 18.06.2025).

24. Никитина, Т. П., Королев, Л. В. Программирование. Основы Python : учебное пособие для СПО / Т. П. Никитина, Л. В. Королев. – Санкт-Петербург : Лань, 2023. – 156 с. – ISBN 978-5-507-45283-5.

25. Петракова, Н. В. Основы HTML. Ч. 1 : учебно-методическое пособие по дисциплине Web-программирование для самостоятельной работы студентов по направлению подготовки 09.03.03 Прикладная информатика / Н. В. Петракова. – Брянск : Брянский ГАУ, 2022. – 50 с. – Б. ц.

26. Пономарчук, Ю. В., Кузнецов, И. В. Программирование на языке Java : учебное пособие / Ю. В. Пономарчук, И. В. Кузнецов. – Хабаровск : ДВГУПС, 2021. – 103 с.

27. PuzzleBot. Бот-платформа для Telegram // PuzzleBot – URL: https://puzzlebot.top/ (дата обращения: 18.06.2025).

28. pyttsx3. Документация // pyttsx3: официальный сайт. – URL: https://pyttsx3.readthedocs.io/ (дата обращения: 17.06.2025).

29. Сергеева, О. А. Программирование на Python : учебно-методическое пособие / О. А. Сергеева. – Кемерово : КемГУ, 2024. – 157 с. – ISBN 978-5-8353-3123-9.

30. Слинкин, А. А., Рамальо, Л. Python – к вершинам мастерства: лаконичное и эффективное программирование / А. А. Слинкин, Л. Рамальо; пер. с англ. А. А. Слинкина. – 2-е изд. – Москва: ДМК Пресс, 2022. – 898 с. – ISBN 978-5-97060-885-2.

31. Снастин, А. В., Северанс, Ч. Python для всех / А. В. Снастин, Ч. Северанс. – Москва: ДМК Пресс, 2022. – 262 с. – ISBN 978-5-93700-104-7.

32. Сьерра, К., Бейтс, Б. Изучаем Java = Head First Java : [перевод с английского] / Кэти Сьерра, Берт Бейтс. – 2-е изд. – Москва : Эксмо, 2016. – 717 с. : ил. – (Мировой компьютерный бестселлер). – Указ.: с. 709-717. – ISBN 978-5-699-54574-2.

33. Telegram Bot API. Документация // Telegram. – URL: https://core.telegram.org/bots/api (дата обращения: 17.06.2025).

34. Федотова, Т. П. Безопасность и конфиденциальность данных в чат-ботах / Т. П. Федотова, В. М. Сушков // Финансовая безопасность – новые горизонты : материалы X Междунар. науч.-практ. конф. (Москва, 19-20 нояб. 2024 г.) / Нац. исслед. ядер. ун-т «МИФИ». – Москва : НИЯУ МИФИ, 2024. – С. 179–187.

35. Фридман, Э. Node.js в действии / Э. Фридман. – Санкт-Петербург : Питер, 2016. – 416 с. – ISBN 978-1-617-29057-2.

36. Хорстманн, К. С. Core Java. Том I: Основы / К. С. Хорстманн. – 10-е изд. – Москва : Питер, 2018. – 928 с. – ISBN 978-0-134-17730-4.

37. Чичулин, А. В. Чат-бот для менеджеров /А.В. Чичулин – Москва : Издательские решения, 2023. – 152 с – ISBN 978-5-0060-0515-0.

38. WhatsApp Business API. Документация // Meta for Developers. – URL: https://developers.facebook.com/docs/whatsapp/ (дата обращения: 17.06.2025).

39. Viber REST API. Документация // Viber Developers Hub. – URL: https://developers.viber.com/docs/api/rest-bot-api/ (дата обращения: 17.06.2025).

40. Титов, А. Н., Тазиева, Р. Ф. Python. Обработка данных : учебно-методическая литература / А. Н. Титов, Р. Ф. Тазиева ; Казанский национальный исследовательский технологический университет. – Казань : Казанский национальный исследовательский технологический университет, 2022. – 104 с. – ISBN 978-5-7882-3171-6.

ПРИЛОЖЕНИЕ А

Листинг кода чат-бота

Чат-бот приемной кампании ЛПИ филиал СФУ

```
from flask import Flask, render_template, request, jsonify
import pyttsx3
from faq_data import faq
import traceback
import threading

app = Flask(__name__)

# Инициализация TTS и блокировка для синхронизации
tts_engine = pyttsx3.init()
tts_lock = threading.Lock()

# Функция поиска с приоритетом по длине совпадения
def get_answer_with_link(question):
    question = question.lower().strip()
    matched = []
    for key, answer in faq.items():
        if key.lower() in question:
            matched.append((len(key), answer))
    if matched:
        matched.sort(key=lambda x: x[0], reverse=True)
        answer = matched[0][1]
        if isinstance(answer, dict):
            # Возвращаем только 'text' и 'link' если есть
            text = answer.get('text', '')
            link = answer.get('link', '')
            # Если нет отдельного link, возвращаем пустую строку
```

```
            return text, link
        else:
            return answer, ''
    return "Извините, я пока не знаю ответа на этот вопрос.", ''

# Функция для озвучивания
def speak_answer(answer):
    with tts_lock:
        tts_engine.say(answer)
        tts_engine.runAndWait()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/ask', methods=['POST'])
def ask():
    try:
        data = request.json
        print("Получен запрос:", data)
        question = data.get("question", "")
        mode = data.get("mode", "text")

        if not question:
            return jsonify({"error": "Пожалуйста, задайте вопрос!"})

        answer, link = get_answer_with_link(question)

        if mode == "text":
            print("Отвечаем текстом:", answer)
```

```
            threading.Thread(target=speak_answer, args=(answer,), daemon=True).start()
            return jsonify({"answer": answer})

        elif mode == "link":
            if not link:
                # fallback на Яндекс поиск
                link = {
                    "url": f"https://yandex.ru/search/?text={question.replace(' ', '+')}",
                    "text": "Поиск в Яндексе"
                }
            print("Отвечаем ссылкой:", link)
            return jsonify({"answer": answer, "link": link})

        else:
            return jsonify({"error": "Неизвестный режим ответа"}), 400

    except Exception as e:
        print(f"Ошибка на сервере: {e}")
        print(traceback.format_exc())
        return jsonify({"error": "Произошла ошибка на сервере."}), 500

if __name__ == '__main__':
    app.run(debug=True, threaded=True)
```

ПРИЛОЖЕНИЕ Б

Листинг кода HTML и CSS

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="UTF-8">
   <title>Чат-бот lpibot</title>
   <style>
      body {
         font-family: 'Times New Roman', Times, serif;
         background-color: #f4f4f9;
         padding: 20px;
         margin: 0;
         font-size: 20px;
         text-align: center;
      }

   .voice-toggle-container button {
         padding: 6px 12px;
         font-size: 14px;
         border-radius: 6px;
         border: 2px solid #F36928;          /* фирменный оранжевый */
         background-color: #F36928;          /* фон кнопки */
         color: #ffffff;                     /* белый текст */
         cursor: pointer;
         user-select: none;
         box-shadow: 0 4px 6px rgba(243, 105, 40, 0.4); /* тень оранжевая */
         transition: background-color 0.3s, box-shadow 0.3s;
      }
```

```
.voice-toggle-container button:hover {
    background-color: #d65920;          /* чуть темнее при наведении */
    box-shadow: 0 6px 8px rgba(214, 89, 32, 0.6);
}

.faq-inline a {
    display: inline-block;
    font-size: 16px;
    text-decoration: none;
    background-color: #e0e0e0;
    color: #0056b3;
    padding: 5px 10px;
    border-radius: 6px;
    font-weight: bold;
    transition: background-color 0.2s;
}

.faq-inline a:hover {
    background-color: #d0d0d0;
}

.modal {
    display: none;
    position: fixed;
    z-index: 2000;
    left: 0;
    top: 0;
    width: 100%;
    height: 100%;
    overflow: auto;
```

```
    background-color: rgba(0, 0, 0, 0.5);
}

.modal-content {
    background-color: #fff;
    margin: 10% auto;
    padding: 20px;
    border-radius: 8px;
    width: 80%;
    max-width: 600px;
    position: relative;
    box-shadow: 0 5px 15px rgba(0,0,0,0.3);
    text-align: left;
}

.modal-content h2 {
    margin-top: 0;
}

.close {
    color: #aaa;
    position: absolute;
    top: 15px;
    right: 20px;
    font-size: 28px;
    font-weight: bold;
    cursor: pointer;
}

.close:hover {
```

```
    color: #000;
}

.header-top {
    display: flex;
    justify-content: space-between; /* Прижать текст влево, FAQ вправо */
    align-items: center;
    gap: 20px;
    margin-bottom: 15px;
}

.logo {
    height: 80px;
    object-fit: contain;
    margin-right: 15px;
}

.header-text {
    font-weight: bold;
    font-size: 18px;
    line-height: 1.2;
    color: #333;
    user-select: none;
    text-align: left;
    display: flex;
    flex-direction: column;
    justify-content: center;
    margin: 0;
    flex-grow: 1; /* Чтобы занять доступное место слева */
}
```

```
.faq-inline {
    margin: 0; /* убираем отступы */
}

.container {
    max-width: 600px;
    margin: 0 auto;
    background-color: #ffffff;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    display: flex;
    flex-direction: column;
    height: 90vh;
}

h1 {
    color: #333;
    margin-top: 0;
    margin-bottom: 10px;
}

#messages {
    margin-top: 20px;
    height: 300px;
    overflow-y: auto;
    border: 1px solid #ddd;
    padding: 10px;
    background-color: #f9f9f9;
```

```
    max-height: 400px;
    flex: 1;
    display: flex;
    flex-direction: column;
}

.message {
    padding: 10px;
    margin: 10px 0;
    border-radius: 12px;
    max-width: 70%;
    display: flex;
    align-items: center;
    word-wrap: break-word;
}

.user {
    background-color: #d3ffd3;
    justify-content: flex-end;
    margin-left: auto;
    text-align: right;
    display: flex;
    align-items: center;
    gap: 10px;
}

.user .avatar {
    width: 40px;
    height: 40px;
    border-radius: 50%;
```

```
    object-fit: cover;
    flex-shrink: 0;
    order: 2;
}

.user .message-text {
    order: 1;
}

.bot {
    background-color: #f0f0f0;
    justify-content: flex-start;
    margin-right: auto;
    text-align: left;
    display: flex;
    align-items: center;
    gap: 10px;
}

.bot .avatar {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    object-fit: cover;
    flex-shrink: 0;
}

input[type="text"] {
    font-size: 18px;
    padding: 10px;
```

```
    border-radius: 4px;
    border: 1px solid #ddd;
    width: 100%;
}

.voice-faq-inline {
    display: flex;
    align-items: center;
    gap: 15px;
}

.voice-faq-inline button {
    padding: 6px 12px;
    font-size: 14px;
    border-radius: 6px;
    border: 2px solid #F36928;
    background-color: #F36928;
    color: #fff;
    cursor: pointer;
    user-select: none;
    box-shadow: 0 4px 6px rgba(243, 105, 40, 0.4);
    transition: background-color 0.3s, box-shadow 0.3s;
}

.voice-faq-inline button:hover {
    background-color: #d65920;
    box-shadow: 0 6px 8px rgba(214, 89, 32, 0.6);
}

.voice-faq-inline a {
```

```
    display: inline-block;
    font-size: 16px;
    text-decoration: none;
    background-color: #e0e0e0;
    color: #0056b3;
    padding: 5px 10px;
    border-radius: 6px;
    font-weight: bold;
    transition: background-color 0.2s;
}

.voice-faq-inline a:hover {
    background-color: #d0d0d0;
}

.button-container {
    display: flex;
    justify-content: center;
    gap: 10px;
}

button {
    padding: 10px 20px;
    background-color: #a19d9d;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    height: 45px;
    display: inline-block;
```

```
    font-size: 16px;
}

#speechButton {
    background-color: #F36928;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    height: 45px;
    width: 45px;
    font-size: 22px;
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 0;
    transition: background-color 0.3s ease;
}

#speechButton:hover {
    background-color: #d95b21; /* чуть темнее при наведении */
}

button:hover {
    background-color: #8f8b8b;
}

#modeButtons {
    display: none;
    margin-top: 10px;
```

```
        }

        #chatInputArea {
            background-color: #fff;
            padding: 10px 20px 20px 20px;
            border-top: 1px solid #ccc;
            margin-top: auto;
            display: flex;
            flex-direction: column;
        }

        .input-row {
            display: flex;
            align-items: center;
            gap: 10px;
        }

        .input-row input[type="text"] {
            flex-grow: 1;
        }

    </style>
</head>
<body>

<!-- Модальное окно -->
<div id="faqModal" class="modal">
    <div class="modal-content">
        <span class="close" onclick="closeFaq()">&times;</span>
        <h2>Часто задаваемые вопросы</h2>
```

```
        <ul>
                <li><strong>Какие документы нужны для поступления?</strong><br>Паспорт, аттестат, СНИЛС и заявление.</li>
                <li><strong>Когда начинается приём документов?</strong><br>С 1 июня текущего года.</li>
                <li><strong>Предоставляется ли общежитие?</strong><br>Да, для всех студентов.</li>
                <li><strong>Какие направления есть?</strong><br>Педагогическое, Психолого-педагогическое, Педагогическое (с двумя профилями), Физическая культура, Педагогика и психология девиантного поведения, Информационные системы и технологии. </li>
        </ul>
    </div>
</div>

<div class="container">
    <div class="header-top">
<div style="display: flex; align-items: center; gap: 15px;">
    <img src="/static/avatar_bot.jpg" alt="Логотип" class="logo">
    <div class="header-text">
        <div>ЛЕСОСИБИРСКИЙ</div>
        <div>ПЕДАГОГИЧЕСКИЙ</div>
        <div>ИНСТИТУТ</div>
    </div>
</div>

<!-- Новый контейнер для кнопки и FAQ -->
<div class="voice-faq-inline">
        <button id="voiceToggleBtn" class="voice-toggle-btn" title="Вкл/Выкл озвучивание">Голос: Вкл</button>
```

```
                <a href="#" onclick="openFaq()" title="Часто задаваемые вопросы">FAQ</a>
    </div>
    </div>

        <h1>Чат-бот приёмной кампании </h1>
        <p>Задай вопрос и выбери, как получить ответ:</p>

        <div id="messages"></div>

        <div id="chatInputArea">
            <div class="input-row">
                    <input type="text" id="question" placeholder="Введите вопрос..." oninput="showModeButtons()">
                 <button id="speechButton" onclick="startSpeechRecognition()" title="Это микрофон">🎤</button>
            </div>

            <div id="modeButtons" class="button-container">
                <button onclick="askQuestion('text')">Ответ получить в окне</button>
                <button onclick="askQuestion('link')">Получить ссылку</button>
            </div>
        </div>

        <div id="error" style="color: red; margin-top: 10px;"></div>
    </div>

    <script>
    let voiceEnabled = true; // озвучивание включено по умолчанию
```

```
const voiceToggleBtn = document.getElementById('voiceToggleBtn');

voiceToggleBtn.addEventListener('click', () => {
    voiceEnabled = !voiceEnabled;
    voiceToggleBtn.textContent = voiceEnabled ? 'Голос: Вкл' : 'Голос: Выкл';
});
let recognition;
if ('webkitSpeechRecognition' in window) {
    recognition = new webkitSpeechRecognition();
    recognition.lang = 'ru-RU';
    recognition.continuous = false;
    recognition.interimResults = false;
    recognition.maxAlternatives = 1;

    recognition.onresult = function(event) {
        const transcript = event.results[0][0].transcript;
        document.getElementById('question').value = transcript;
        showModeButtons();
    };

    recognition.onerror = function(event) {
         document.getElementById('error').innerText = 'Ошибка при распознавании речи. Попробуйте еще раз.';
    };
}

function startSpeechRecognition() {
    if (recognition) {
        recognition.start();
    }
```

```
}

function showModeButtons() {
    const question = document.getElementById('question').value.trim();
    const modeButtons = document.getElementById('modeButtons');
    modeButtons.style.display = question ? 'flex' : 'none';
}

let voices = [];

window.speechSynthesis.onvoiceschanged = () => {
    voices = window.speechSynthesis.getVoices();
};

function speakText(text) {
    if (!voiceEnabled) return;  // Если голос выключен — ничего не озвучиваем

    // остальной код озвучивания...
    if (!voices.length) {
        voices = window.speechSynthesis.getVoices();
    }

    const utterance = new SpeechSynthesisUtterance(text);
    // ...
    window.speechSynthesis.speak(utterance);
    }
    if (!voices.length) {
        voices = window.speechSynthesis.getVoices();
    }
```

```
    const utterance = new SpeechSynthesisUtterance(text);
    const selectedVoice = voices.find(voice =>
        voice.lang.startsWith('ru') &&
        (voice.name.toLowerCase().includes('google') ||
        voice.name.toLowerCase().includes('microsoft') ||
        voice.name.toLowerCase().includes('anna') ||
        voice.name.toLowerCase().includes('irina'))
    );

    if (selectedVoice) {
        utterance.voice = selectedVoice;
    }

    utterance.pitch = 1.2;
    utterance.rate = 0.9;
    window.speechSynthesis.speak(utterance);

async function askQuestion(mode) {
    const questionInput = document.getElementById('question');
    const question = questionInput.value.trim();
    if (!question) {
        alert('Пожалуйста, введите вопрос.');
        return;
    }

    const messagesDiv = document.getElementById('messages');

    const userMsg = document.createElement('div');
    userMsg.classList.add('message', 'user');
```

```
const userText = document.createElement('span');
userText.classList.add('message-text');
userText.textContent = "ВЫ: " + question;

const userAvatar = document.createElement('img');
userAvatar.classList.add('avatar');
userAvatar.src = '/static/avatar_user.jpg';

userMsg.appendChild(userText);
userMsg.appendChild(userAvatar);
messagesDiv.appendChild(userMsg);

questionInput.value = '';
showModeButtons();

try {
    const response = await fetch('/ask', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ question: question, mode: mode })
    });

    const data = await response.json();

    if (data.error) {
        alert(data.error);
        return;
    }

    const botMsg = document.createElement('div');
```

```
            botMsg.classList.add('message', 'bot');
            const botAvatar = document.createElement('img');
            botAvatar.classList.add('avatar');
            botAvatar.src = '/static/avatar_bot.jpg';
            const answerSpan = document.createElement('span');
    if (mode === 'text') {
        // Показываем полный текст (answer)
          const answerText = (typeof data.answer === 'object' && data.answer.text) ? data.answer.text : data.answer;
        answerSpan.innerHTML = "lpibot: " + answerText;
        speakText(answerText);
    } else if (mode === 'link') {
    let linkHref = '';
    let linkText = '';
    if (data.link) {
        if (typeof data.link === 'object') {
        linkHref = data.link.url;
        linkText = data.link.text || data.link.url;
        } else if (typeof data.link === 'string') {
        linkHref = data.link;
        linkText = data.link;
        }
    }
        answerSpan.innerHTML = `lpibot: <a href="${linkHref}" target="_blank" rel="noopener noreferrer">${linkText}</a>`;
    speakText(linkText); // если нужно озвучить
    }
            botMsg.appendChild(botAvatar);
            botMsg.appendChild(answerSpan);
            messagesDiv.appendChild(botMsg);
```

```
        messagesDiv.scrollTop = messagesDiv.scrollHeight;
    } catch (err) {
        alert('Ошибка при соединении с сервером.');
        console.error(err);
    }
}
function openFaq() {
    document.getElementById('faqModal').style.display = 'block';
}
function closeFaq() {
    document.getElementById('faqModal').style.display = 'none';
}
window.onclick = function(event) {
    const modal = document.getElementById('faqModal');
    if (event.target === modal) {
        modal.style.display = "none";
    }
}
// === Добавленная функция для кликабельной ссылки в ответе FAQ ===
function sendFollowup(question) {
    document.getElementById('question').value = question;
    showModeButtons();
    askQuestion('link'); // здесь 'link' чтобы получить ссылку, как ты хочешь
}
</script>
</body>
</html>
```