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

ДОПУСТИТЬ К ЗАЩИТЕ

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

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка Telegram-бота «Помощник врача»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-576.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ Т.А. Макаровских

Автор работы,
студент группы КЭ-433
_____ И.А. Мантров

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

Челябинск, 2024 г.

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

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

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-433
Мантрову Ивану Алексеевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. №764-13/12) Разработка Telegram-бота «Помощник врача».
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Срини Д. Практическое руководство по разработке чат-интерфейсов: пер. с англ.: М.Райтман. – М.: ДМК Пресс, 2019. – 340 с.
3.2. Telegram. [Электронный ресурс] URL: https://core.telegram.org/ (дата обращения: 29.01.2024 г.).
3.3. Документация pyTelegramBotAPI. [Электронный ресурс] URL: https://pypi.org/project/pyTelegramBotAPI/ (дата обращения: 29.01.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести обзор научной литературы и предметной области.
4.2. Изучить существующие аналоги и разработать необходимые требования к системе.
4.3. Выполнить разработку бота.
4.4. Разработать тестовые наборы и провести тестирование системы.

5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

Задание принял к исполнению И.А. Мантров

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

Современный этап развития информационных технологий и цифровизации сферы здравоохранения предъявляет к разработкам особые требования, направленные на совершенствование функционирования медицинских учреждений и повышение качества предоставляемой медицинской помощи. Несмотря на значительные достижения в области медицины и технологий, многие люди по-прежнему сталкиваются с проблемами доступа к качественной медицинской помощи. В связи с этим возникает потребность в разработке инновационных решений, способных упростить и улучшить процессы диагностики, лечения и профилактики различных заболеваний.

Одним из перспективных направлений в данной области является создание Telegram-бота «Помощник врача». Данный бот представляет собой сервис, интегрированный в мессенджер Telegram. Он способен анализировать симптомы, введенные пользователем, и предлагать возможные диагнозы, а также рекомендации по обращению к специалистам. Кроме того, бот может предоставить алгоритмы действий для оказания первой помощи.

Постановка задачи:

Цель настоящей выпускной квалификационной работы заключается в разработке Telegram-бота «Помощник врача», а также анализе его функциональности, преимуществ и недостатков. Для достижения данной цели предполагается решение следующих задач:

1) обзор научной литературы;
2) анализ требований и потребностей пользователей;
3) проектирование архитектуры и интерфейса бота;
4) разработка системы на языке Python с использованием библиотеки telebot;
5) тестирование и отладка бота.

Структура и содержание работы:

Данная работа включает в себя введение, две главы, заключение и список литературы. Общий объем работы составляет 50 страниц, а список литературы включает 18 источников.

В первой главе «Обзор и анализ средств разработки чат-бота» проведен анализ существующих аналогов, их преимущества и недостатки. А также используемых технологий.

Во второй главе «Требования к проектируемой системе» представлены функциональные и нефункциональные требования к разрабатываемому боту, диаграмма вариантов использования, спецификация основных вариантов использования и общее описание архитектуры.

В третьей главе «Разработка чат-бота» описаны средства и методы реализации чат-бота. Описана работа с API Telegram, процесса получения HTTP API токена от BotFather и его значение для аутентификации бота. Интерфейс пользователя, включая дизайн и функциональность интерфейса. Иллюстрация основных элементов интерфейса. Описание базы знаний, ее структура и содержания.

В четвертой главе описан процесс тестирования разработанного чат-бота различными методами, такими как: модульное и функциональное тестирование, а также тестирование методом белого ящика.

В приложении А показаны спецификации вариантов использования бота. В приложении Б представлен листинг реализации работы Telegram-бота. А в приложении В представлена диаграмма вариантов использования чат-бота.

1. ОБЗОР И АНАЛИЗ СРЕДСТВ РАЗРАБОТКИ ЧАТ-БОТА

1.1. Общие понятия и классификации чат-ботов

К сегодняшнему дню чат-боты стали неотъемлемой частью различных сфер деятельности, заменяя или дополняя человека. Сейчас компании часто прибегают к использованию виртуальных собеседников. В общем, чат-бот – это программное обеспечение, выполняющее автоматические задачи и общающееся с пользователями через Интернет [1]. Например, пользователь может задать боту вопрос, и он ответит или выполнит соответствующее действие. Взаимодействие с чат-ботом происходит в формате, аналогичном обмену мгновенными сообщениями.

В то время как боты могут имитировать человеческий разговор, они отличаются от человека-оператора тем, что могут работать автономно. Боты обрабатывают вопросы пользователей на естественном языке и предоставляют ответы на основе заранее определенных сценариев и методов машинного обучения. Они используют базу данных для формирования ответов.

Чат-боты имеют различные типы и классификации [2].

1. Примитивные боты предоставляют базовый набор функций и обычно используются для предоставления информации или ответов на часто задаваемые вопросы. Они обычно работают по простым сценариям и не требуют сложной обработки естественного языка.

2. Саморазвивающиеся боты обучаются на основе данных и опыта взаимодействия с пользователями. Они способны улучшать свою работу, адаптируясь к новым ситуациям и обучаясь на основе обратной связи.

3. Кнопочные боты используются для простого взаимодействия с пользователем с помощью кнопок или меню. Они часто используются в мессенджерах, где доступ к клавиатуре ограничен, и предоставляют быстрый способ выбора опций.

4. Текстовые боты обрабатывают естественный язык и предоставляют ответы на основе анализа вводимого текста пользователя. Они используют различные методы и алгоритмы для понимания запросов и предоставления соответствующих ответов.

5. Коммуникационные боты предназначены для общения с пользователем и проведения диалогов. Они могут использоваться для различных целей, от развлекательных бесед до выполнения операций в бизнес-процессах.

6. Функциональные боты предназначены для выполнения конкретных функций или задач, таких как бронирование билетов, заказ товаров или получение информации о погоде. Они обычно интегрируются с другими сервисами или системами для выполнения этих задач.

Классификация чат-ботов может быть полезной для определения целей и требований к разрабатываемому боту, а также для выбора наиболее подходящих методов и технологий для его реализации.

Каждый тип имеет свои особенности и предназначен для различных задач, от простых консультаций до выполнения сложных функций, таких как покупки и оплата услуг.

1.2. Выбор мессенджера

Мессенджеры и социальные сети стали одними из самых популярных приложений, которыми пользуются люди всех возрастов [3]. Согласно статистики Tgstat в Telegram доля пользователей младше 18 лет, пользователей от 18 до 24 лет, от 25 до 34 лет, от 35 до 44 лет, а также 45 лет и старше распределены относительно равномерно, но пользователей возрастом от 25 до 34 лет традиционно больше всех – почти 30% [4]. Это разнообразие возрастных групп подчеркивает универсальность интерфейса и функций Telegram, которые находят отклик у широкого круга пользователей. Поэтому Telegram можно назвать «мессенджером не только для прогрессивных» распределение по возрасту показывает, что это массовый продукт.

Так, мессенджер активно используется как школьниками, так и пенсионерами, делая его по-настоящему массовым продуктом (рисунок 1) [4].

Рисунок 1 – Распределение по возрасту

Telegram имеет несколько преимуществ перед другими мессенджерами. Для сравнения были выбраны Viber, WhatsApp и WeChat.

Сравнивая Telegram и Viber, хочется отметить, что Telegram имеет широкое международное распространение и активную аудиторию, особенно в странах СНГ и Европы. Это делает его предпочтительным выбором для многих пользователей в этих регионах. В то же время, WhatsApp обладает большим комьюнити пользователей по всему миру, что делает его чрезвычайно популярным. Одним из главных преимуществ Telegram является его открытая платформа для разработчиков. Telegram предоставляет открытый API и дружественную среду для разработчиков, что упрощает создание и интеграцию ботов, в отличие от Viber и WhatsApp. Это особенно важно для компаний, которые хотят использовать ботов для автоматизации своих

бизнес-процессов и улучшения взаимодействия с клиентами.

Telegram также обеспечивает высокий уровень защиты данных и приватности пользователей, включая возможность шифрования конфиденциальных сообщений и использование двухфакторной аутентификации (2FA). Это может быть важным фактором для компаний, уделяющих большое внимание безопасности и конфиденциальности данных. В отличие от Telegram, WhatsApp хоть и предлагает конечное шифрование для защиты конфиденциальности сообщений, но не всегда обеспечивает такую же прозрачность в системе безопасности.

WeChat – один из самых популярных мессенджеров в Китае, который предлагает широкий спектр функций, включая платежи, заказ еды, бронирование билетов и многое другое. Он также интегрирован с другими сервисами, такими как социальные сети и онлайн-магазины. В этом плане Telegram уступает китайскому мессенджеру, поскольку в Telegram только недавно были реализованы возможности создания игр и добавлены платежи. Однако WeChat ограничен в своем международном распространении из-за строгих правил контроля и цензуры, применяемых китайским правительством, что делает его менее привлекательным для пользователей за пределами Китая.

Viber, несмотря на свою ограниченную популярность в некоторых регионах, также имеет свои уникальные преимущества. Он поддерживает создание ботов для бизнеса и обеспечивает высокую функциональность в этой сфере. Однако его возможности интеграции и предложений для бизнеса часто уступают Telegram, особенно в контексте международного использования, масштабируемости и открытости документации для разработчиков.

В таблице 1 представлены преимущества и недостатки рассмотренных мессенджеров.

Таблица 1 – Преимущества и недостатки мессенджеров [5]

Мессенджер	Преимущества	Недостатки
Telegram	Открытость платформы для разработчиков и API	Не имеет такой широкой базы пользователей, как WhatsApp
	Богатый набор функциональных возможностей (игры, платежи, медиа-контент и др.)	Возможность быстрой распространения фейковых новостей из-за открытости каналов и групп
	Высокий уровень защиты данных и приватности пользователей	
WhatsApp	Огромная база пользователей по всему миру	Отсутствие возможности добавления ботов и других развлекательных функций
	Конечное шифрование для защиты конфиденциальности сообщений	Система безопасности может быть не столь прозрачной, как у Telegram
Viber	Поддержка создания ботов для бизнеса	Менее распространен в некоторых регионах по сравнению с другими мессенджерами
		Относительно ограниченные интеграции и возможности для бизнеса
WeChat	Один из самых популярных мессенджеров в Китае	Ограниченный доступ за пределами Китая, что делает его менее привлекательным для международного рынка
	Широкий спектр функций, включая платежи, заказ еды, бронирование билетов и многое другое	Ограничения в связи с контролем и цензурой со стороны китайского правительства

1.3. Преимущества чат-ботов и выбор ПО для разработки

Преимущества чат-ботов для пользователей

Чат-боты обеспечивают мгновенную обратную связь и позволяют пользователям избежать долгого ожидания ответа. Они сразу предоставляют ответы на вопросы, что очень удобно для простых запросов от пользователя.

Чат-боты также помогают пользователям быстро находить нужную информацию, упрощая процесс ее поиска. В отличие от людей, они способны быстро обработать большой объем данных и предоставить ответы за считанные секунды.

Преимущества чат-ботов для организации

Чат-боты представляют собой инструмент для установления персонального взаимодействия организации с пользователем [5]. Они обеспечивают доступность, оперативность и круглосуточное обслуживания, а также способствуют созданию позитивного пользовательского опыта. Подбор подходящей личности чат-бота, основанный на брендовой идентичности организации и предпочтениях аудитории, играет важную роль в этом процессе.

Выбор программного обеспечения для разработки чат-бота

В данный момент существует множество технологий для разработки чат-бот. В Telegram также существует несколько инструментов для разработки ботов.

1. BotFather. Это официальный бот от Telegram, который предоставляет возможность создавать и настраивать собственных ботов. С помощью BotFather можно легко регистрировать новых ботов и управлять ими через простой интерфейс команд.

2. Telegram Bot API. Это API, предоставляемое Telegram, позволяет разработчикам создавать и управлять чат-ботами для Telegram с помощью языков программирования, таких как Python, Go, Java.

3. Python и библиотеки. Для разработки чат-ботов для Telegram можно использовать различные библиотеки Python, такие как python-telegram-bot или Telepot, которые предоставляют удобные инструменты для взаимодействия с API Telegram [6].

4. Для размещения Telegram-бота на облачном сервере был выбран бесплатный облачный сервис, такой как Replit. Эта платформа предоставляет простой способ развертывания приложений с минимальной настройкой и поддержкой интеграции с GitHub для автоматического размещения.

5. Мониторинг состояния и работы сервера происходит на платформе UptimeRobot. Она поддерживаются различные виды мониторинга, такие как HTTP(s), Ping, Port, и Keyword Платформа позволяет настраивать

уведомления о сбоях и восстановлении через различные каналы, такие как электронная почта, SMS, Telegram

Разработка чат-бота в такой среде, как Python, особенно эффективна благодаря широкому сообществу и обширной документации, что позволяет быстро находить решения возникающих проблем и узнавать о лучших практиках. Использование среды разработки, такой как PyCharm, дополнительно упрощает процесс разработки за счет интегрированных инструментов для отладки, управления зависимостями и работы с кодом.

1.4. Анализ существующих чат-ботов

В данный момент в Telegram существуют аналоги разрабатываемого бота. У каждого из них есть преимущества и недостатки. Подобно человеку, каждый чат-бот индивидуален и имеет свои уникальные черты. В этом разделе рассмотрены три бота предоставляющих информацию по первой помощи.

«Джгут 2.0 – Fast AID bot 2.0» [7]

«Джгут 2.0» (рисунок 2) предоставляет пользователю всю информацию о первой помощи, которая может потребоваться в экстренных ситуациях. Бот пошагово показывает алгоритм действий, в зависимости от сценария, который выбрал пользователь, а также предлагает пользователю видео инструкцию. Такой алгоритм поможет неподготовленному человеку оказать первую помощь. Также в боте предусмотрен режим тренировки.

Преимуществом бота является большой список информации и быстрый отклик на действия пользователя, а также наличие видео инструкций значительно упрощает понимание и выполнение действий, необходимых для оказания первой помощи.

Недостатком является отсутствие русскоязычного интерфейса. Этот аспект существенно ограничивает доступность и полезность бота для русскоязычной аудитории.

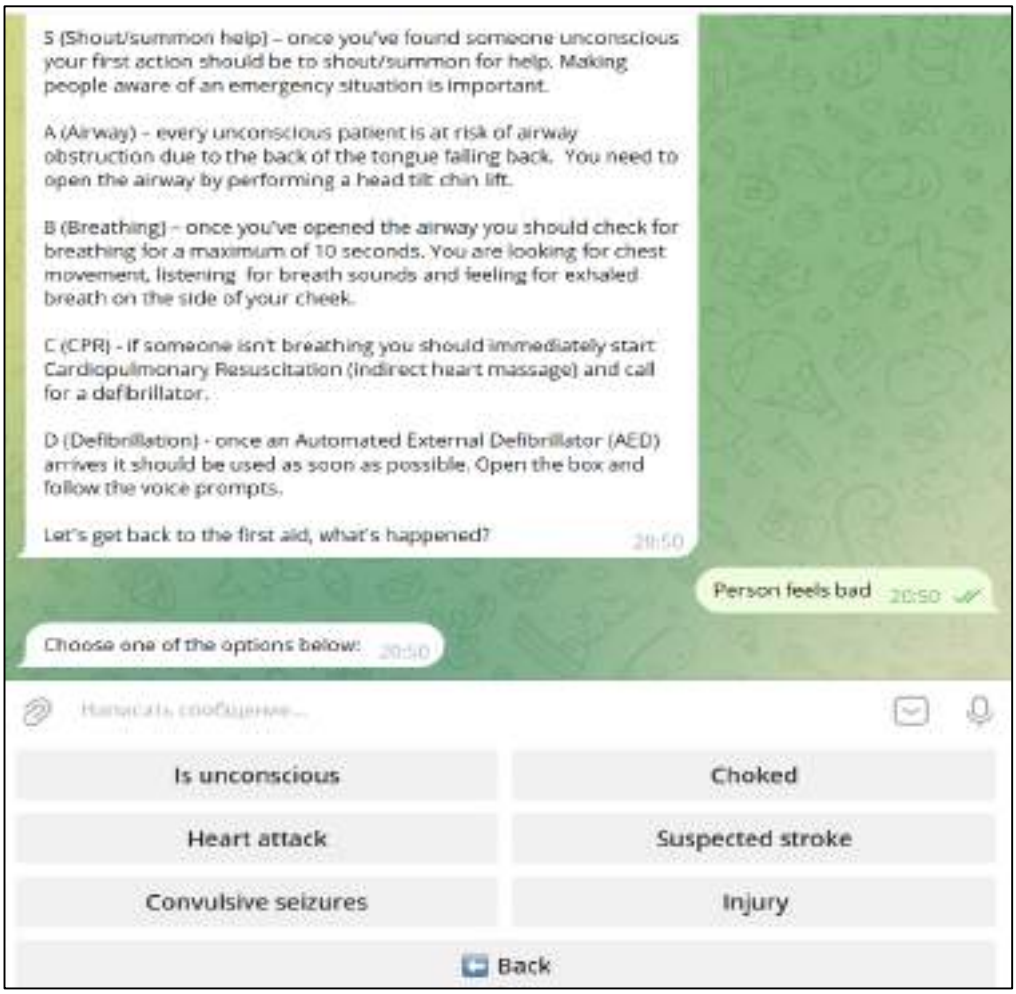

Рисунок 2 – Бот «Джгут 2.0»

«Здоробот» [7]

«Здоробот» предлагает своим пользователям всестороннюю помощь. Бот (рисунок 3) может анализировать симптомы, помочь диагностировать заболевание, предложить нужные лекарства, указав их стоимость, а также найти ближайшие аптеки, где можно их приобрести. В случае необходимости, запишет на прием к врачу в одну из проверенных клиник.

Преимуществом является большой объем предоставляемой информации и быстрота реагирования на действия пользователя.

Недостатком является отсутствие заявленной возможности записи к врачу. При попытке записаться, бот предлагает ввести команды, которые в нем не предусмотрены.

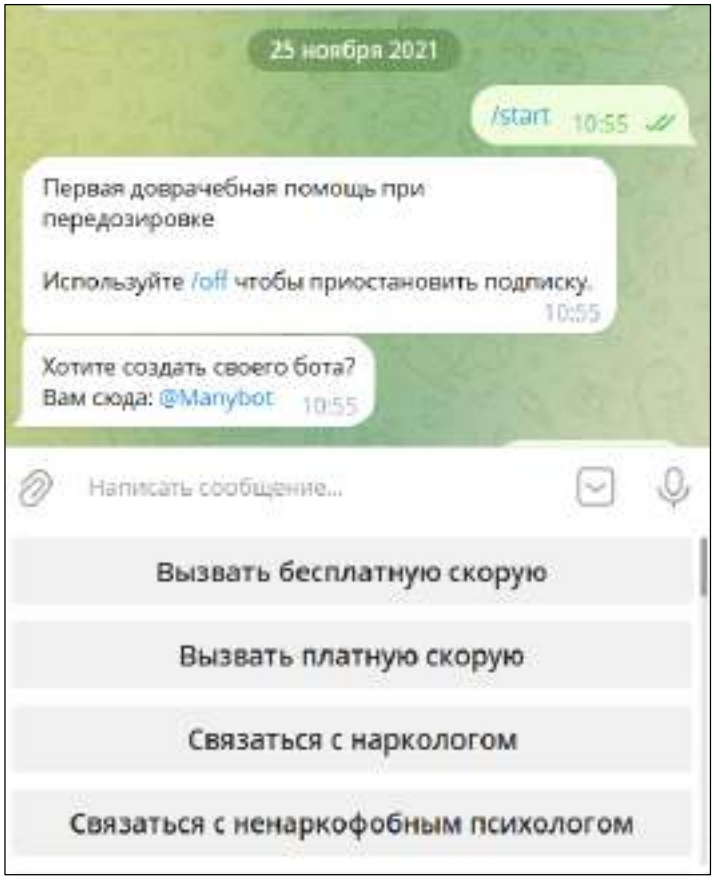

Рисунок 3 – Бот «Здоробот»

«Академия первой помощи»

Бот «Академии первой помощи» (рисунок 4), который предоставляет пользователю ссылки на видео с платформы YouTube при введении пользователем тега его медицинской проблемы, например кровь, рана, перелом, ушиб, укус, повязка, аптечка и т.д. Также бот содержит кнопки при нажатии, на которые бот дает ссылку на обучающее видео или обучающий сайт.

Преимущества бота поиск, по ключевым словам, тегам. Пользователь может сразу ввести запрос.

У этого бота есть и свои недостатки. Основным недостатком является отсутствие в боте собственной полезной информации. Все ответы бота ведут на сторонние ресурсы, что может быть неудобно для пользователя и занимать много времени. Это означает, что пользователи вынуждены поки-

дать интерфейс бота для просмотра видео или чтения инструкций, что может снизить оперативность и эффективность оказания первой помощи.

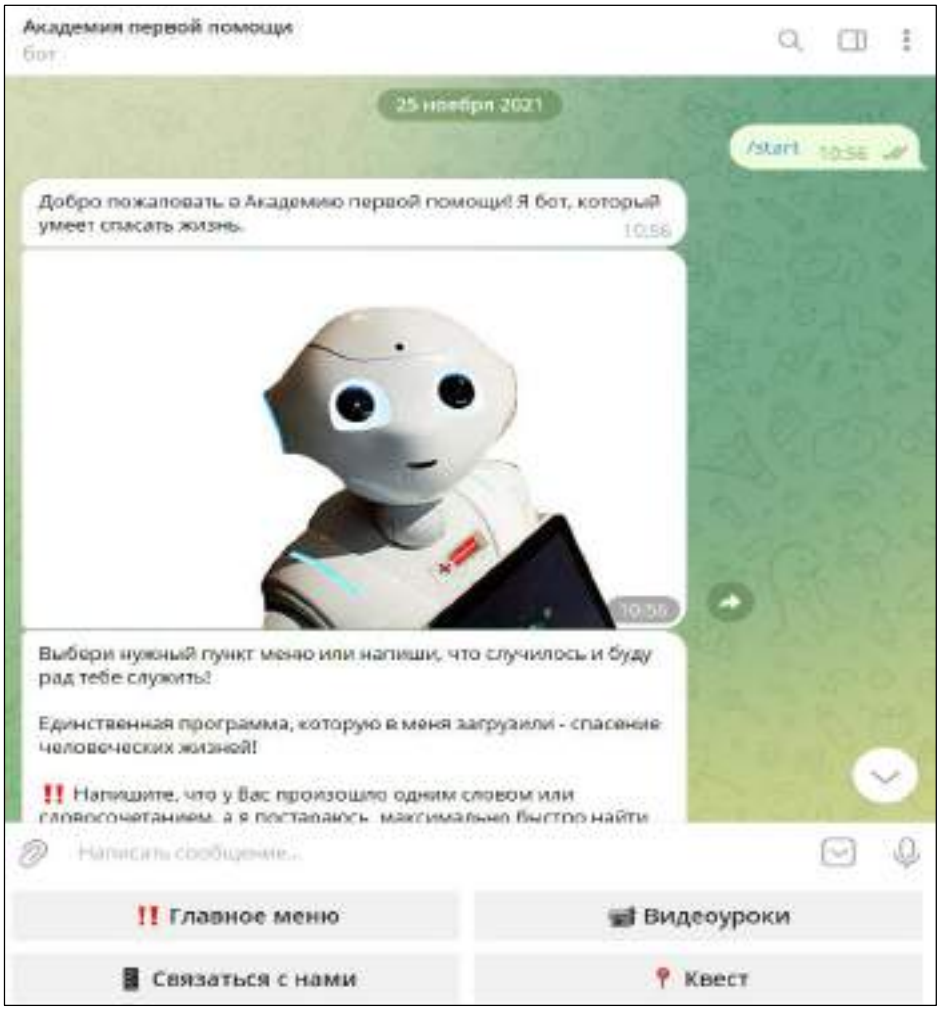

Рисунок 4 – Бот «Академия первой помощи»

Анализ аналогов, таких как Джгут 2.0, Здоробот и Академия первой помощи, показал, что каждый из этих ботов имеет свои уникальные преимущества и недостатки. Джгут 2.0 предоставляет подробные алгоритмы первой помощи и тренировочный режим, что делает его полезным инструментом для обучения и практики. Однако отсутствие русскоязычного интерфейса может ограничить его использование среди русскоязычных пользователей. Здоробот предлагает всестороннюю помощь, включая анализ симптомов и рекомендации по лекарствам, а также возможность записи на прием к врачу, что делает его очень полезным в повседневной жизни. Однако отсутствие полной функциональности записи на прием к врачу снижает его

эффективность. Академия первой помощи предоставляет быстрый доступ к обучающим видео, но недостаток собственной информации и зависимость от сторонних ресурсов могут быть неудобными для пользователей.

Таким образом, путем анализа и сопоставления различных аналогов с разрабатываемым ботом, а также изучения преимуществ и недостатков Telegram чат-ботов по оказанию первой помощи и их функционала позволило сформировать таблицу 2.

Таблица 2 – Сравнение похожих функций ботов

Функции	**Джгут 2.0**	**Здоробот**	**Академия первой помощи**
Первая помощь	Да	Нет	Нет
Алгоритмы действий	Да	Нет	Нет
Тренировочный режим	Да	Нет	Нет
Анализ симптомов	Нет	Да	Нет
Диагностирование	Нет	Да	Нет
Предложение лекарств	Нет	Да	Нет
Поиск ближайших аптек	Нет	Да	Нет
Запись на прием к врачу	Нет	Да	Нет
Предоставление ссылок на видео	Нет	Да	Нет
Поиск по ключевым словам, тегам	Нет	Нет	Да
Предоставление полезной информации	Нет	Нет	Нет

Вывод по первой главе

В первой главе были рассмотрены общие понятия и классификации чат-ботов. Описаны возможности каждого типа ботов. Также был проведен анализ мессенджеров таких как Telegram, WhatsApp, Viber и WeChat. Анализ показал, что оптимальным выбором для разработки бота является платформа Telegram, так как он обладает открытой документацией для разработчиков, богатому функционалу и высокой защите данных. Также в данной главе было описано выбранное ПО Python и библиотека python-telegram-bot, а для развертывания облачный сервис Replit. Помимо этого, были рассмотрены аналоги разрабатываемой системы. Также были выделены их преимущества, недостатки и функциональные возможности. Анализ аналогов показал, что разрабатываемый бот обладает уникальными функциями, и объединяет в себе функциональные возможности рассмотренных аналогов.

2. ТРЕБОВАНИЯ К ПРОЕКТИРУЕМОЙ СИСТЕМЕ

2.1. Функциональные и нефункциональные требования к системе

Функциональные требования – это описание того, что система должна делать [8]. Они определяют функции, сервисы или операции, которые система должна обеспечивать, включая ввод, вывод и обработку данных. Эти требования четко определяют поведение системы и ее возможности. Для данной системы функциональные требования таковы:

1) авторизация через Telegram;
2) возможность связаться в техподдержкой;
3) взаимодействие с пользователем через кнопки и команды;
4) получение информации о различных симптомах, лечения;
5) помощь в экстренных ситуациях – предоставление алгоритма по оказанию первой помощи;
6) возможность устанавливать напоминания;
7) просмотр погодных условия по заданному городу;
8) возможность расчета ИМТ.
9) бот должен предоставлять инструкцию по использованию бота.

Нефункциональные требования – это характеристики или ограничения, которые определяют способность системы выполнять свои функции, но не описывают эти функции напрямую [8]. Они определяют качественные аспекты системы, такие как производительность, надежность, безопасность, удобство использования и т.д. Нефункциональные требования описывают, как система должна работать, а не что она должна делать. Для данной системы нефункциональные требования таковы:

1) отклик на запрос пользователя без задержек;
2) бот должен иметь адаптивный дизайн, который обеспечивает удобство использования на различных устройствах и разрешениях экрана;
3) система должна работать на различных ОС и платформах;
4) система должна быть простой в использовании и понятной;

5) бот должен быть доступен круглосуточно с минимальными перерывами на техническое обслуживание.

Эти требования определяют основу для разработки чат-бота, обеспечивая его функциональность и удобство использования для пользователя.

2.2. Диаграмма вариантов использования

Диаграмма вариантов использования – это тип диаграммы унифицированного языка моделирования (UML), которая представляет взаимодействие между субъектами (пользователями или внешними системами) и рассматриваемой системой для достижения конкретных целей. Он обеспечивает представление о функциональности системы, иллюстрируя способы взаимодействия пользователей с ней [9].

В рамках проекта чат-бота были идентифицированы и разработаны, предъявляемые к чат-боту, требования и варианты его использования. Диаграмма вариантов использования представлены на рисунке 1 приложения В.

Основные актеры, взаимодействующие с системой с системой, взаимодействует пользователь, которым является любой человек, зарегистрированный в мессенджере Telegram и использующий функционал чат-бота, например, студент.

Основное назначение диаграммы вариантов использования – предоставить, с одной стороны, достаточно формальное, с другой стороны, достаточно удобное, и, с третьей стороны, достаточно универсальное средство, позволяющее до некоторой степени снизить риск расхождений в толковании спецификаций [10]. Таким образом, диаграмма вариантов использования служит ключевым элементом в планировании и разработке сложных систем, предоставляя ясное и структурированное видение взаимодействия между пользователями и системой.

В дополнение к ранее сформулированным требованиям, в проекте были созданы диаграммы классов, деятельности и последовательности, которые помогут более точно определить и визуализировать работу чат-бота. Эти

диаграммы позволяют улучшить понимание структуры системы и взаимодействий между ее компонентами.

Диаграмма классов (рисунок 5) описывает типы объектов системы и различного рода статические отношения, которые существуют между ними. На диаграммах классов отображаются также свойства классов, операции классов и ограничения, которые накладываются на связи между объектами [11].

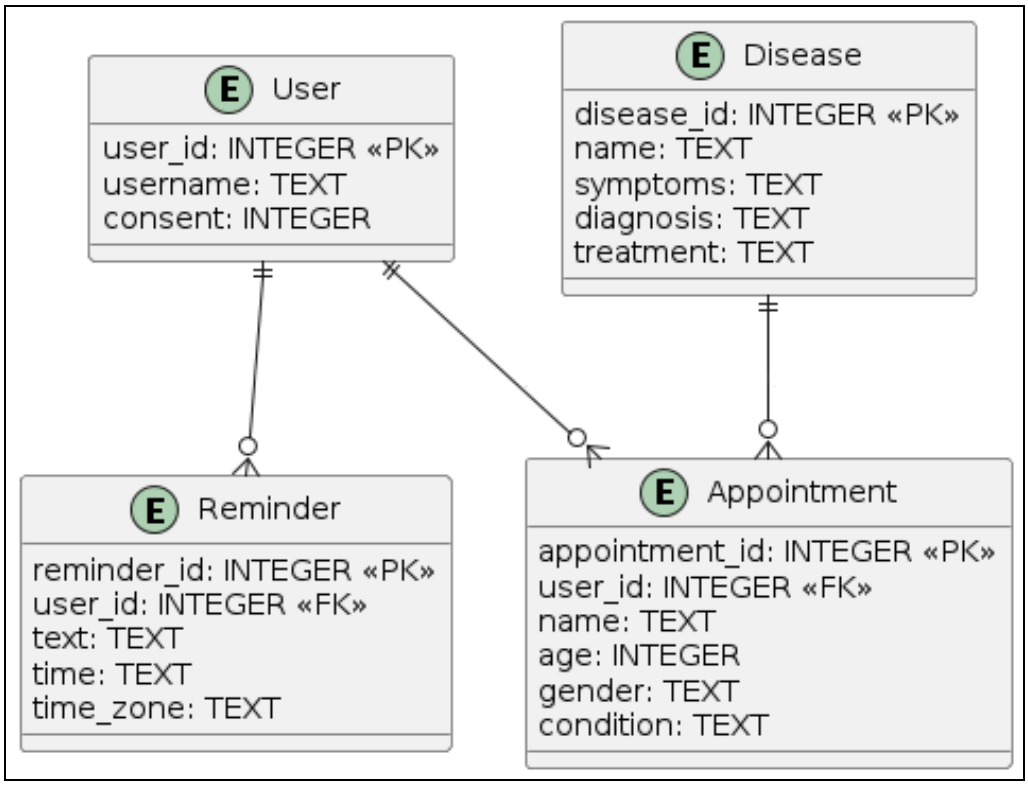

Рисунок 5 – Диаграмма классов

2.3. Спецификация и описание архитектуры системы

Спецификация основных вариантов использования

UML – стандарта для спецификации прецедента не существует [12]. Однако широко используется шаблон, применяющийся в данной главе. Есть и более сложные шаблоны, но при моделировании прецедентов лучше всего придерживаться максимальной простоты, так как простые шаблоны проще понимать и при необходимости вносить изменения. В приложении А приведены спецификации основных вариантов использования.

Общее описание архитектуры системы

Разрабатываемый чат-бот представляет собой комплексную систему взаимодействия с пользователем в мессенджере Telegram. Основной функционал чат-бота заключается в предоставлении пользователям удобного интерфейса для выполнения различных действий и получения информации, связанных с медицинскими услугами и первой помощью. Пользователь взаимодействует с ботом через Telegram, используя кнопки интерфейса, предоставляемые ботом. Кнопки представляют собой команды или действия, которые пользователь может выбрать для получения информации или выполнения конкретных операций.

Система чат-бота основана на заранее прописанной логике и алгоритмах, которые определяют реакцию бота на различные команды и запросы пользователя. При получении команды от пользователя через интерфейс кнопок, бот анализирует выбор пользователя и формирует соответствующий ответ на основе встроенных алгоритмов. Каждая кнопка интерфейса ассоциирована с определенной командой или действием, которые заранее прописаны в системе чат-бота. При выборе определенной кнопки пользователем, бот активирует соответствующий алгоритм обработки запроса и формирует ответное сообщение на основе предварительно определенных правил.

Для хранения информации о пользователях, напоминаниях, медицинских записях и других данных используется база данных SQLite. База данных обеспечивает надежное хранение и быстрый доступ к необходимой информации. Чат-бот интегрирован с различными API для получения актуальной информации, такой как погода (OpenWeatherMap) и местоположение больниц (Google Maps).

В качестве облачного хостинга используется Replit. Он позволяет запускать и обеспечивать работу бота в круглосуточном режиме. Это особенно важно для сервисов, предоставляющих такие функции, как напоминания о приеме лекарств, которые должны работать без перебоев. Также

Repplit упрощает работу с кодом, делаю ее доступной с любого устройства.

Для обеспечения стабильной работы бота и отслеживания его доступности бота используется сервис мониторинга UptimeRobot. В случае обнаружения проблем система уведомляет разработчиков для оперативного устранения неисправностей.

Выводы по второй главе

В ходе анализа требований были определены ключевые функциональные и нефункциональные требования к разрабатываемой системе, что позволило установить основу для дальнейшего проектирования. Были разработаны диаграммы вариантов использования, которые иллюстрируют взаимодействие пользователя с системой и возможные сценарии использования. Спецификации вариантов использования предоставили детальное описание шагов для достижения конкретных целей. Также была создана диаграмма классов, которая отразила структуру системы. Эти этапы работы обеспечили четкое понимание архитектуры системы и способствовали созданию модульной и функциональной структуры чат-бота.

3. РАЗРАБОТКА ЧАТ-БОТА

В соответствии с требованиями, к разрабатываемой системе которые были определены в предыдущей главе, для реализации системы чат-бота использовался язык Python 3. Написание программного кода и его отладки осуществлялась в среде разработки PyCharm Community Edition.

Процесс разработки бота начинается с создания учетной записи бота в мессенджере Telegram с использованием административного бота – BotFather, который показан на рисунке 6.

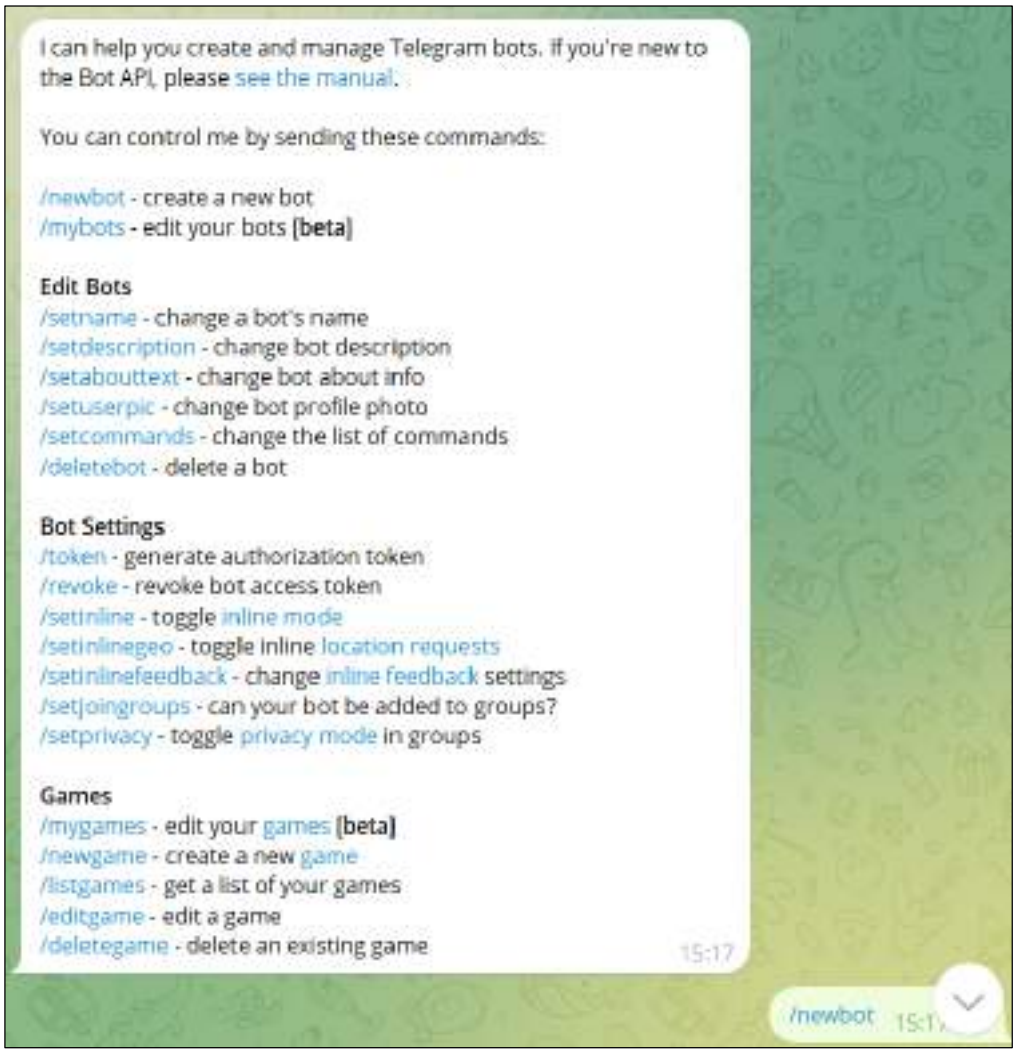

Рисунок 6 – Административный бот Telegram

3.1. Используемые API в чат-боте

После создания, BotFather предоставляет HTTP API токен для осуществления запросов к серверу Telegram. В данном контексте токен – это

строка, которая аутентифицирует вашего бота (не вашу учетную запись) в bot API. У каждого бота есть уникальный токен, который также может быть отозван в любое время через BotFather.

Получить токен так же просто, как связаться с BotFather, выполнить команду `/newbot` и следовать инструкциям, пока вам не будет предоставлен новый токен [13]. Он будет выглядеть так: `4839574812:AAFD39kkdpWt3ywyRZergyOLMaJhac60qc`.

Запросы к серверу могут быть выполнены как с использованием метода GET, так и POST. Многие из методов API требуют дополнительных параметров. Например, для метода `sendMessage`, который отвечает за отправку сообщений, требуется указать идентификатор чата (`chat_id`) и текст сообщения. Передача этих параметров может осуществляться через URL-строку запроса или в формате «`application/x-www-form-urlencoded`» и «`application-json`», за исключением операций, связанных с загрузкой файлов [14].

Для работы с запросами в мессенджере Telegram и обработки входящих сообщений использовалась библиотека telebot. Эта библиотека предназначена для создания и управления ботами в Telegram, обеспечивая простой и эффективный способ взаимодействия с Telegram Bot API. Telebot обладает широким набором функций, позволяющих ботам обрабатывать различные типы сообщений, реагировать на события и взаимодействовать с пользователями. Она реализует методы для отправки сообщений, управления клавиатурами и многие другие функции, необходимые для эффективной работы бота в Telegram.

Также в боте используется OpenWeatherMap API которое предоставляет доступ к актуальной информации о погоде для любого города. Этот API позволяет получить данные о текущей погоде. Пример использования API представлен в листинге 1.

Листинг 1 – Пример использования API

```
def send_weather_info(city):
    api_key = 'YOUR_OPENWEATHERMAP_API_KEY'
```

```
    url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&ap-
pid={api_key}&units=metric&lang=ru'
    response = requests.get(url)
    data = response.json()
    if data['cod'] == 200:
weather_info=f"Погодав{city}:\nТемпература:{data['main']['temp']}°C\nПогода
: {data['weather'][0]['description']}"
        return weather_info
```

Еще одно API используемое в боте это Google Maps API. Оно предоставляет возможность работать с картами Google, включая получение картографических данных, создание маршрутов, поиск мест и многое другое. В данном проекте используется функциональность генерации ссылок на карты. В боте оно используется для поиска ближайших медицинских учреждений. Пример использования представлен в листинге 2.

Листинг 2 – Использование API Google Maps

```
def get_hospital_map_link():
    return "https://www.google.com/maps/search/hospital"
```

3.2. Интерфейс пользователя

Интерфейс пользователя составляет первый слой взаимодействия между чат-ботом и пользователем. Он спроектирован с учетом необходимости обеспечить интуитивно понятное и удобное взаимодействие пользователей без специальной подготовки. Интерфейс предоставляет возможность ввода запросов с использованием кнопок с предварительно предложенными вариантами и получения ответов в форме диалога. Пример интерфейса приведен на рисунке 7.

Разработанный интерфейс ориентирован на обеспечение максимальной простоты и доступности в использовании, что позволяет пользователям легко ориентироваться и взаимодействовать с ботом. Он представляет собой ключевой элемент в обеспечении позитивного опыта использования бота и способствует повышению его эффективности в решении поставленных задач.

Представленный интерфейс демонстрирует основные элементы взаимодействия, включая вариативные кнопки для ввода запросов и последующего получения информации. Расположение элементов интерфейса и их дизайн универсальны для всех ботов мессенджера Telegram.

Таким образом, интерфейс пользователя представляет собой важное средство обеспечения удобства и эффективности взаимодействия с чат-ботом, что является ключевым фактором успешной реализации и популяризации данного программного решения.

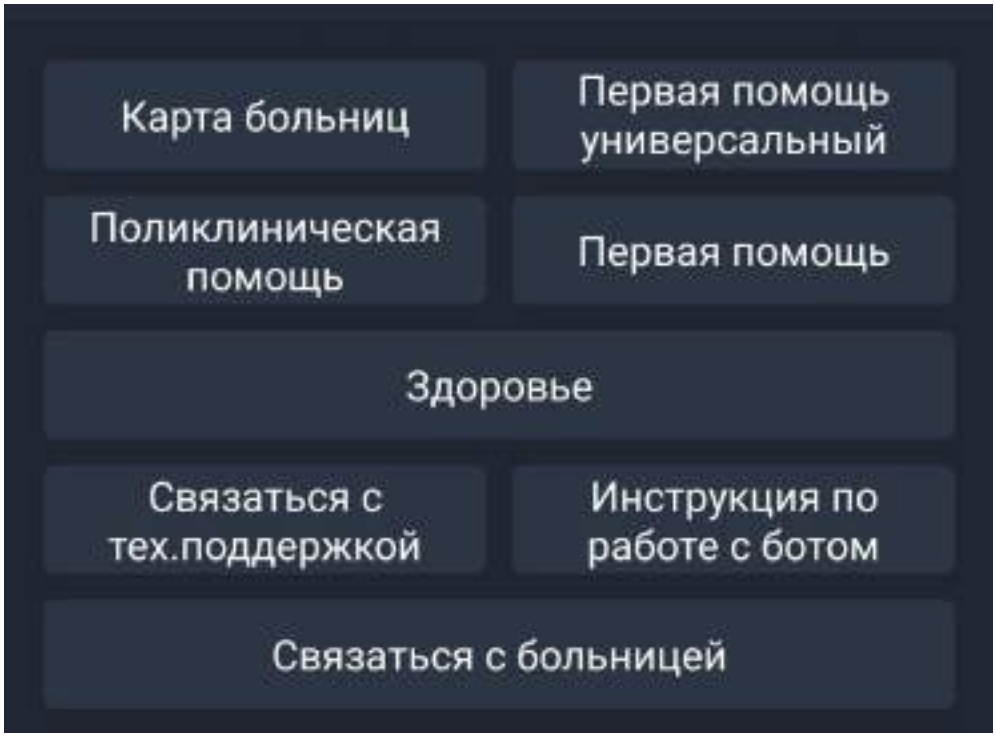

Рисунок 7 – Интерфейс бота

Чат-боты Telegram предоставляют два основных типа клавиатур для взаимодействия с пользователем: стандартную клавиатуру (`ReplyKeyboard-Markup`) которая заменяет стандартную клавиатуру устройства пользователя и обеспечивает простой способ отправки команд и текстовых сообщений непосредственно боту. Клавиатуры такого типа идеально подходят для сценариев, где пользователю необходимо выбирать из предопределенных опций, что уменьшает вероятность ошибок и упрощает взаимодей-

ствие. Кнопки на такой клавиатуре могут быть настроены на отправку специфических текстовых сообщений, что делает ее особенно удобной для сценариев, требующих быстрых и стандартных ответов. И инлайн-клавиатуру (`InlineKeyboardMarkup`) которая в отличие от стандартной, отображается прямо в теле сообщения и может содержать кнопки, которые выполняют обратные вызовы, переносят пользователя по URL-адресам или запускают специфические функции бота. Такая клавиатура превращает сообщение чат-бота в интерактивный интерфейс, где каждая кнопка может представлять различные действия, такие как голосования, формы обратной связи или навигация по функциям бота. Это расширяет возможности взаимодействия и делает общение с ботом более динамичным и функциональным.

В данном чат-боте используются несколько типов клавиатур.

1. `ReplyKeyboardMarkup` (рисунок 7). Используется для создания основной клавиатуры, которая отображается в чате и включает кнопки с вариантами выбора для пользователя. На листинге 3 представлено создание основной клавиатуры [15].

2. `InlineKeyboardMarkup` (рисунок 8). Используется для создания встроенных клавиатур, которые привязаны к конкретным сообщениям и могут обрабатывать нажатия на кнопки с помощью callback функций. На листинге 4 представлено создание встроенной клавиатуры [15].

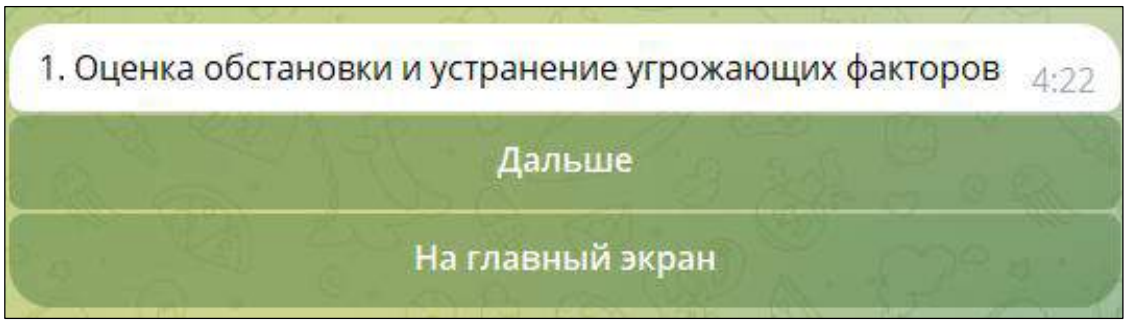

Рисунок 8 – Пример встроенной клавиатуры

Листинг 3 – Создание основной клавиатуры

```
def create_main_menu():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Инструкция по работе')
    return markup
```

Листинг 4 – Создание встроеной клавиатуры

```
def create_inline_keyboard():
    markup = types.InlineKeyboardMarkup()
    markup.add(types.InlineKeyboardButton("Кнопка1", callback_data='btn1'))
    return markup
```

Для удаления текущей клавиатуры используется `ReplyKeyboardRemove`.

Клавиатура может быть настроены для выполнения различных действий и предоставления разнообразного взаимодействия с пользователем. Использование клавиатур значительно упрощает навигацию по функциям бота и повышает удобство его использования.

3.3. База данных

База данных чат-бота является важным элементом его архитектуры. Данные для базы данных собираются из различных надежных источников, включая медицинские журналы и официальные рекомендации здравоохранительных организаций. База данных содержит набор медицинских данных, включая симптомы, диагнозы и лечение.

Для хранения данных, используемых в чат-боте, была использована реляционная база данных SQLite. Схема базы данных представлена на рисунке 9. Она включает в себя таблицы с их атрибутами и связями между ними, что обеспечивает целостность данных и позволяет эффективно их обрабатывать.

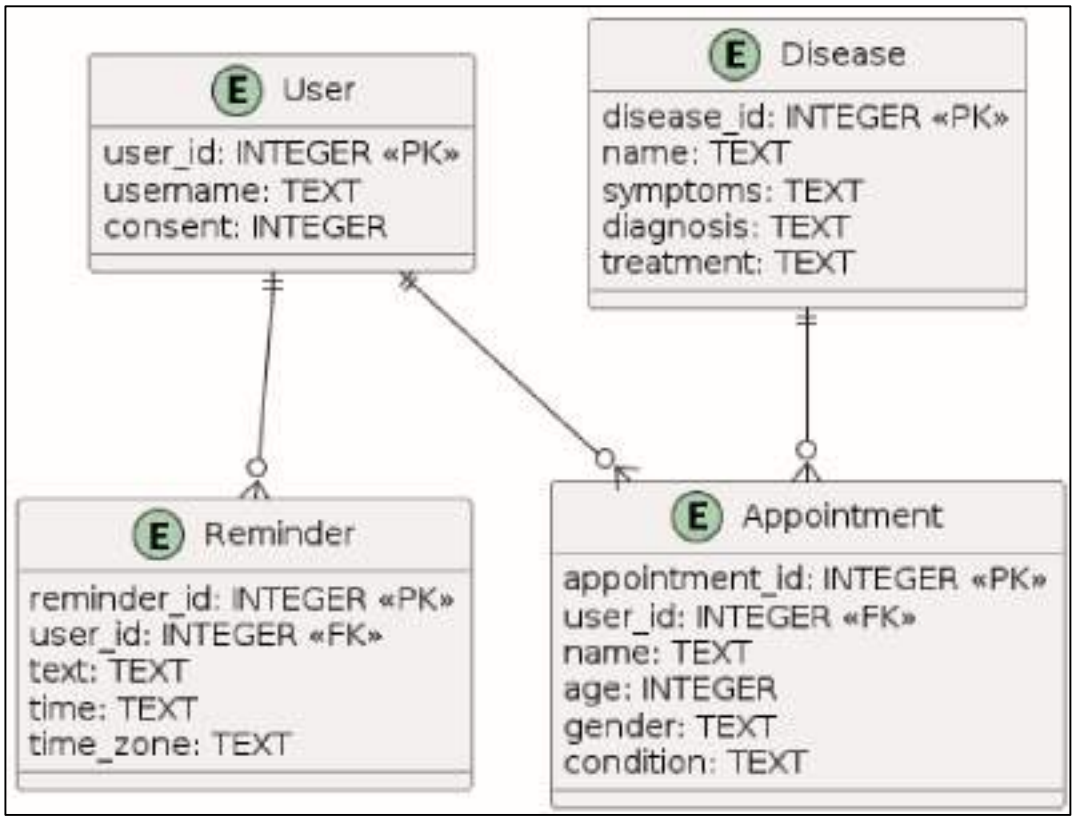

Рисунок 9 – Схема базы данных

Эта база данных позволяет хранить информацию о пользователях, напоминаниях и других взаимодействиях с ботом. Основные таблицы включают:

1) users – хранит информацию о пользователях, такие как идентификатор пользователя и настройки;

2) reminders – хранит идентификатор напоминания, идентификатор пользователя, текст напоминания, время напоминания и часовой пояс;

3) diseases – содержит идентификатор заболевания, а также информацию о заболеваниях, включая симптомы, диагностику и лечение;

4) appointment – хранит идентификатор записи, идентификатор пользователя, имя, возраст и пол пользователя, а также жалобу пользователя.

Этот набор данных организован и структурирован таким образом, чтобы обеспечить эффективный доступ к необходимой информации в про-

цессе формирования ответов на запросы конечных пользователей. Текстовая информация, которую выводит бот при ответе пользователю пишется в коде программы в идентификаторе `message.chat.id`.

В процессе взаимодействия с пользователем чат-бот использует базу знаний для формирования ответов на запросы. Например, при получении запроса о симптомах гриппа, бот может предоставить информацию о распространенных симптомах, методах домашнего лечения и советах по профилактике. Интерфейс чат-бота позволит пользователям легко и быстро получать необходимую медицинскую информацию, делая общение с ботом максимально удобным и полезным.

Как было отмечено ранее, база знаний является ключевым элементом архитектуры чат-бота, обеспечивающим надежное и эффективное хранение, организацию и доступ к медицинской информации, необходимой для его функционирования и взаимодействия с пользователем.

3.4. Дополнительные функции

Одной из ключевых функций бота является интеграция с внешними сервисами для предоставления следующих возможностей:

1. Погода. Пользователь может запросить текущую погоду в своем городе. Для этого бот отправляет запрос к OpenWeatherMap API и выводит пользователю актуальную информацию о погоде.

2. Напоминания. Пользователь может устанавливать напоминания о приеме лекарств или других событиях. Бот сохраняет напоминания в базе данных и отправляет уведомления в заданное время. Также пользователь может посмотреть и удалить установленные ранее напоминания.

3. Расчет индекса массы тела (ИМТ). Пользователь может ввести свои данные (рост и вес), и бот рассчитает индекс массы тела.

4. Удаление напоминаний. Пользователь может удалить установленные напоминания.

Помимо интеграции с внешними сервисами в боте реализована возможность записи на прием ко врачу. Пользователь может записаться на прием к врачу, указав свои данные (ФИО, возраст, пол, состояние). Бот сохраняет эту информацию в базе данных и помогает организовать запись. Также реализована возможность при необходимости связаться с технической поддержкой или ознакомиться с инструкцией по работе с чат-ботом.

Выводы по третьей главе

В третьей главе подробно рассмотрены аспекты разработки чат-бота для платформы Telegram. Создание и настройки бота через BotFather. Ключевой аспект – база знаний бота, которая включает информацию о заболеваниях и инструкциях по первой помощи. Бот интегрируется с внешними сервисами для предоставления дополнительных возможностей: получение текущей погоды через OpenWeatherMap API, установка и управление напоминаниями, расчет индекса массы тела, запись на прием к врачу и предоставление инструкций по оказанию первой помощи.

4. ТЕСТИРОВАНИЕ ЧАТ-БОТА

Для обеспечения качественной работы чат-бота «Помощник врача» были проведены различные виды тестирования, включая функциональное тестирование, тестирование методом белого ящика, нагрузочное тестирование и модульное тестирование. В данном разделе описаны методы и результаты этих тестов.

4.1. Функциональное и модульное тестирование

Функциональное тестирование – это процесс проверки программного обеспечения для подтверждения соответствия его функциональных требований [16]. В таблице 3 приведен протокол тестирования некоторых аспектов работы системы.

Таблица 3 – Функциональное тестирование чат-бота

№	Аспект работы	Действия	Результат	Тест пройден?
1.	Стартовое сообщение	Ввести команду /start.	Бот отправляет приветственное сообщение и выводит клавиатуру с опциями	Да
2.	Вывод алгоритма первой помощи (шаг 1)	Нажать кнопку «Первая помощь».	Бот выводит первый шаг алгоритма первой помощи и кнопку «Дальше».	Да
3.	Переход к следующему шагу алгоритма	Нажать кнопку «Дальше».	Бот выводит следующий шаг алгоритма и кнопку «Дальше».	Да
4.	Расчет ИМТ	Нажать кнопку Рассчитать ИМТ	Бот выводит запрос доп. информации и после рассчитывает ИМТ.	Да
5.	Запрос общей поликлинической помощи	Нажать кнопку «Поликлиническая помощь».	Бот запрашивает у пользователя, какая информация необходима, и выводит инструкции.	Да
6.	Получение информации о симптомах	Ввести команду «Симптомы»	Бот предоставляет возможные симптомы	Да
7.	Запись ко врачу	Выбрать запись ко врачу	Бот запрашивает ФИО, возраст, пол, и заболевание и записывает пользователя в базу	Да

Окончание таблицы 3

№	Аспект работы	Действия	Результат	Тест пройден?
8.	Вывод информации о ближайших больницах	Нажать кнопку «Поиск больниц».	Бот предоставляет ссылку на карту с ближайшими больницами.	Да
9.	Обработка запросов на первую помощь	Ввести команду «Первая помощь» и следовать алгоритму.	Бот корректно выводит инструкции	Да
10.	Отображение кнопок с командами	Нажать кнопку «Общая помощь»	Бот корректно отображает клавиатуру с командами.	Да
11.	Вывод справочной информации	Ввести команду «Инструкция».	Бот выводит список всех доступных команд и их описание.	Да
12.	Тестирование производительности	Вводить команды в быстром темпе для проверки отклика.	Бот обрабатывает запросы без задержек и сбоев, поддерживая быструю реакцию.	Да

Модульное тестирование

Модульное тестирование включает проверку отдельных модулей и компонентов системы. Это позволяет выявить ошибки на ранних этапах разработки и обеспечить высокое качество кода [17].

Были написаны и выполнены тесты для проверки отдельных функций, таких как отправка сообщений, обработка команд, взаимодействие с базой данных и API.

Для модульного тестирования использовались библиотеки unittest и pytest. Эти инструменты позволили автоматизировать процесс тестирования и повысить его эффективность. В листинге 5 представлен пример модульного тестирования для функции расчета ИМТ.

Листинг 5 – Тестирование функции расчета ИМТ

```
import unittest
from main import calculate_bmi
class TestBMI(unittest.TestCase):
    def test_calculate_bmi(self):
        self.assertAlmostEqual(calculate_bmi(180, 75), 23.15, places=2)
        self.assertAlmostEqual(calculate_bmi(160, 60), 23.44, places=2)
if __name__ == '__main__':
    unittest.main()
```

4.2. Тестирование методом белого ящика

Термин «белый ящик» означает, что при разработке тестовых случаев тестировщик использует любые доступные сведения о внутренней структуре или коде [18].

Тестирование методом белого ящика включает в себя функциональное и модульное тестирование, но не ограничивается ими. Оно также включает в себя проверку внутренней структуры и логики кода. Этот метод позволяет выявить ошибки в алгоритмах, логике работы функций и методах.

Были проведены тесты для проверки корректности обработки различных команд, таких как вывод симптомов диагнозов и методов лечения, уведомлений, погоды и расчета индекса массы тела. Каждый тест включал проверку правильности выполнения условий и циклов, обработки входных данных и генерации ответов.

Произведено тестирование правильности формирования и отправки запросов к OpenWeatherMap API для получения информации о погоде. Также проверялась обработка ответов от API и корректность вывода информации пользователю. Также была проверена корректность работа Google-Map API, и корректный вывод карты больницами.

Выводы по четвертой главе

В этой главе в рамках тестирования были проведены функциональное и модульное тестирование, а также тестирование методом белого ящика которое позволило выявить и устранить ошибки которые не были выявлены на предыдущих этапах тестирования. Тесты показали, что система работает корректно, выполняя все заявленные функции, включая корректную работу подключенный API. Модульное тестирование и тестирование методом белого ящика помогли выявить и устранить ошибки в коде на ранних этапах разработки.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработан многофункциональный Telegram-бот «Помощник врача». При этом были решены следующие задачи.

1. Проведен обзор научной литературы.
2. Проанализированы требования и потребности пользователей.
3. Спроектирована архитектура и интерфейс бота.
4. Разработана система на языке Python с использованием библиотеки telebot.
5. Проведено тестирование и отладка бота.

Разработанный бот предоставляет пользователям широкий спектр услуг, включая консультации по заболеваниям и оказанию первой помощи, расчет ИМТ, напоминания о приеме лекарств и актуальную информацию о погоде. Интеграция с внешними API и создание базы данных обеспечили высокую функциональность и персонализированный подход к каждому пользователю. Разработка и развертывание бота на облачной платформе, а также реализация системы мониторинга обеспечили стабильную и надежную работу системы. Проведенное тестирование подтвердило, что бот полностью соответствует заявленным требованиям и готов к использованию. Разработанный проект продемонстрировал успешное применение современных технологий и инструментов для разработки чат-ботов.

ЛИТЕРАТУРА

1. Чат-бот. [Электронный ресурс] URL: https://bigenc.ru/c/chat-bot-a074a8 (дата обращения: 25.03.2024 г.).

2. Чат-бот: виды классификация предназначение. [Электронный ресурс] URL: https://grandsoft.tech/ru/blog/CHat-boty-vidy-klassifikaciya-prednaznachenie/ (дата обращения: 14.01.2024 г.).

3. Исследование аудитории Telegram 2023. [Электронный ресурс] URL: https://tgstat.ru/research-2023 (дата обращения: 04.02.2024 г.).

4. Исследование аудитории Telegram 2023. [Электронный ресурс] URL: https://tgstat.ru/research-2023 (дата обращения: 04.02.2024 г.).

5. Козориз А.В. Чат-боты как новый инструмент организации взаимодействия с клиентом. // Экономика: вчера, сегодня, завтра, 2019 – Том 9 – № 10А – С. 639 – 648.

6. Гочияева М.Д., Батачаева М.Х. Обзор языков программирования для разработки Телеграм-бота // Тенденции развития науки и образования. – 2023. – С. 24 – 26.

7. Обзор 5 популярных телеграм ботов в медицине. [Электронный ресурс] URL: https://medispark.io/ru/razrabotka-telegramm-botov/obzor-5-populyarnyh-telegramm-botov-v-meditsine/ (дата обращения: 10.06.2023 г.).

8. Определение требований. [Электронный ресурс] URL: https://cs.petrsu.ru/~kulakov/courses/requirements/lectures/2.property (дата обращения: 11.04.2024 г.).

9. Диаграммы вариантов использования | Унифицированный язык моделирования (UML). [Электронный ресурс] URL: https://www.geeksforgeeks.org/use-case-diagram/ (дата обращения: 25.03.2024 г.)

10. Иванов Д.Ю., Новиков Ф.А. Моделирование на UML. Унифицированный язык моделирования UML: учебное пособие. // Изд-во Политехнического университета, 2011. – 11 с.

11. Арлоу Д., Нейштадт А. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. 2-е издание. Пер. с англ. – СПб: Символ-Плюс, 2007. – 624 с.

12. Диаграммы UML. [Электронный ресурс]. URL: https://www.zabudsky.ru/Lecture_12_13_diag_UML (дата обращения: 31.05.2024 г.).

13. From BotFather to «Hello World». [Электронный ресурс] URL: https://core.telegram.org/bots/tutorial (дата обращения: 03.03.2024 г.).

14. Функции Telegram-бота. [Электронный ресурс] URL: https://core.telegram.org/bots/features (дата обращения: 21.04.2024 г.).

15. Байкова К.Д., Медведева Т.А. Анализ и разработка функционального Telegram-бота. // Молодой исследователь Дона, 2021. – № 6(33). – 6 с.

16. Функциональное тестирование. [Электронный ресурс] URL: https://www.qa-book.cloud/vidy-testirovaniya/funkcionalnoe-testirovanie (дата обращения: 25.03.2024 г.).

17. Модульное тестирование. [Электронный ресурс] URL: https://intuit.ru/studies/courses/1040/209/lecture/5394 (дата обращения: 19.05.2024 г.).

18. Тестирование информационных систем методов «белого ящика». [Электронный ресурс] URL: https://revolution.allbest.ru/programming/00566063_0.html (дата обращения: 08.06.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–5.

Таблица 1 – Спецификация ВИ при получении информации о симптомах и диагнозах

UseCase: получение информации о симптомах и диагнозах
ID: 1.
Аннотация: пользователь запрашивает у бота информацию о медицинских симптомах и возможных диагнозах.
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловия: пользователь инициировал диалог с чат-ботом.
Основной поток: прецедент начинается когда пользователь выбирает команду болезни
Постусловия: пользователь получает информацию о возможных диагнозах и рекомендациях по дальнейшим действиям.
Альтернативные потоки: нет

Таблица 2 – Спецификация ВИ получении первой помощи

UseCase: получение первую помощь
ID: 2.
Аннотация: пользователь запрашивает помощь у бота, нажимая кнопку «Помощь».
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловия: пользователь находится в основном меню бота.
Основной поток: система отображает список доступных видов помощи.
Постусловия: пользователь получает доступ к списку видов помощи от бота.
Альтернативные потоки: нет

Таблица 3 – Спецификация ВИ: Напоминания

UseCase: напоминание
ID: 3
Аннотация: пользователь устанавливает напоминание через чат-бот.
Главные актеры: пользователь
Второстепенные актеры: нет
Предусловия: пользователь находится в меню бота и выбирает опцию «Напоминания».
Основной поток: пользователь вводит текст напоминания и указывает время. Система сохраняет напоминание и отправляет уведомление в заданное время.
Постусловия: напоминание установлено, и пользователю будет отправлено сообщение в указанное время.
Альтернативные потоки: если пользователь вводит неверное время или текст, система выводит сообщение об ошибке и предлагает повторить ввод.

Таблица 4 – Спецификация ВИ: Просмотр погоды

UseCase: погода
ID: 4
Аннотация: пользователь запрашивает текущую погоду в своем городе.
Главные актеры: пользователь
Второстепенные актеры: OpenWeatherMap API
Предусловия: пользователь находится в меню бота и выбирает опцию «Погода».
Основной поток: пользователь вводит название города, система отправляет запрос к API погоды и выводит текущую информацию о погоде в указанном городе.
Постусловия: пользователь получает актуальную информацию о погоде в выбранном городе.
Альтернативные потоки: если город введен неверно, система выводит сообщение об ошибке и предлагает повторить ввод.

Таблица 5 – Спецификация ВИ: Расчет ИМТ

UseCase: расчет индекса массы тела (ИМТ)
ID: 5
Аннотация: пользователь вводит свои параметры (рост и вес), и бот рассчитывает индекс массы тела (ИМТ), после чего предоставляет соответствующую информацию о состоянии веса пользователя.
Главные актеры: пользователь
Второстепенные актеры:
Предусловия:
Основной поток: пользователь вводит рост, вес, система производит расчет и выводит информацию о ИМТ пользователю.
Постусловия: пользователь получит информацию о своем ИМТ и рекомендации относительно состояния веса.
Альтернативные потоки: если данные введены неверно, система выводит сообщение об ошибке и предлагает повторить ввод.

Приложение Б. Листинг реализации Telegram чат-бота

В листинге 1 представлена реализация работы Telegram-бота

Листинг 1 – Реализация работы Telegram-бота

```
import os
import telebot
from telebot import types
import sqlite3
import time
import threading
from datetime import datetime
import pytz
import requests
import re
from PIL import Image
# Получение токена вашего бота из переменных окружения для безопасности
token = 'TOKEN_API'
bot = telebot.TeleBot(ПА)
# Подключение к базе данных SQLite
conn = sqlite3.connect('medical_bot.db', check_same_thread=False)
cursor = conn.cursor()
# Создание таблицы reminders, если она не существует
cursor.execute('''CREATE TABLE IF NOT EXISTS reminders
                  (id INTEGER PRIMARY KEY AUTOINCREMENT,
                   user_id INTEGER,
                   text TEXT,
                   time TEXT,
                   time_zone TEXT)''')
# Создание таблицы users для хранения согласия пользователей, ID и никнейма
cursor.execute('''CREATE TABLE IF NOT EXISTS users
                  (user_id INTEGER PRIMARY KEY,
                   consent INTEGER DEFAULT 0,
                   username TEXT)''')
conn.commit()
# Создание таблицы appointments с нужными столбцами
cursor.execute('''CREATE TABLE IF NOT EXISTS appointments
                  (id INTEGER PRIMARY KEY AUTOINCREMENT,
                   user_id INTEGER,
                   name TEXT,
                   age INTEGER,
                   gender TEXT,
                   condition TEXT)''')
conn.commit()
# Функция для создания основного меню
def main_menu():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Карта больниц', 'Первая помощь универсальный алгоритм')
    markup.row('Поликлиническая помощь', 'Первая помощь')
    markup.row('Здоровье')
    markup.row('Связаться с тех.поддержкой', 'Инструкция по работе с бо-
том')
    markup.row('Связаться с больницей')
    return markup
# Функция для создания меню "Здоровье"
def health_menu():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Напоминания', 'Погода')
    markup.row('Рассчитать ИМТ')
    markup.row('Назад')
    return markup
# Обработчик команды /start
```

```
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.send_message(message.chat.id, "Выберите опцию:", re-
ply_markup=main_menu())

def user_gave_consent(user_id):
    cursor.execute("SELECT consent FROM users WHERE user_id = ?",
(user_id,))
    result = cursor.fetchone()
    return result and result[0] == 1
# Функция для обработки напоминаний
def handle_reminders(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Установить напоминание', 'Просмотреть напоминания', 'Уда-
лить напоминание')
    markup.row('Назад', 'На главный экран')
    bot.send_message(message.chat.id, "Выберите опцию:", re-
ply_markup=markup)
# Обработчик для установки напоминания
@bot.message_handler(func=lambda message: message.text == "Установить напо-
минание")
def set_reminder(message):
    msg = bot.send_message(message.chat.id, "Введите текст напоминания:",
reply_markup=cancel_markup())
    bot.register_next_step_handler(msg, get_reminder_text)
# Функция для получения текста напоминания от пользователя
def get_reminder_text(message):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Установка напоминания отме-
нена.", reply_markup=main_menu())
        return
    user_id = message.chat.id
    reminder_text = message.text
    msg = bot.send_message(user_id, "Введите время напоминания в формате
ЧЧ:ММ (например, 14:30):", reply_markup=cancel_markup())
    bot.register_next_step_handler(msg, get_reminder_time, reminder_text)
# Функция для получения времени напоминания от пользователя
def get_reminder_time(message, reminder_text):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Установка напоминания отме-
нена.", reply_markup=main_menu())
        return
    user_id = message.chat.id
    reminder_time = message.text
    try:
        datetime.strptime(reminder_time, "%H:%M")
        msg = bot.send_message(user_id, "Введите ваш часовой пояс
(например, 'Europe/Moscow'):", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, save_reminder, reminder_text,
reminder_time)
    except ValueError:
          msg = bot.send_message(user_id, "Неверный формат времени. Введите
время в формате ЧЧ:ММ (например, 14:30):", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, get_reminder_time, re-
minder_text)
# Функция для сохранения напоминания в базе данных
def save_reminder(message, reminder_text, reminder_time):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Установка напоминания отме-
нена.", reply_markup=main_menu())
        return
    user_id = message.chat.id
```

```
    time_zone = message.text
    try:
        # Проверка правильности часового пояса
        pytz.timezone(time_zone)
        cursor.execute("INSERT INTO reminders (user_id, text, time,
time_zone) VALUES (?, ?, ?, ?)",
                       (user_id, reminder_text, reminder_time, time_zone))
        conn.commit()
        bot.send_message(user_id, "Напоминание установлено!", re-
ply_markup=health_menu())
    except pytz.UnknownTimeZoneError:
        msg = bot.send_message(user_id, "Неверный формат. Введите ваш часо-
вой пояс (например, 'Europe/Moscow'):", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, save_reminder, reminder_text,
reminder_time)
# Функция для создания меню с кнопкой "Отмена"
def cancel_markup():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True, one_time_key-
board=True)
    markup.add('Отмена')
    return markup
# Обработчик для просмотра напоминаний
@bot.message_handler(func=lambda message: message.text == "Просмотреть
напоминания")
def view_reminders(message):
    cursor.execute("SELECT id, text, time, time_zone FROM reminders WHERE
user_id = ?", (message.chat.id,))
    reminders = cursor.fetchall()
    if reminders:
        response = "Ваши напоминания:\n"
        for reminder in reminders:
            response += f"{reminder[0]}. {reminder[1]} - {reminder[2]}
({reminder[3]})\n"
    else:
        response = "У вас нет напоминаний."
    bot.send_message(message.chat.id, response, reply_markup=health_menu())
# Обработчик для удаления напоминаний
@bot.message_handler(func=lambda message: message.text == "Удалить напоми-
нание")
def delete_reminder(message):
    cursor.execute("SELECT id, text, time FROM reminders WHERE user_id =
?", (message.chat.id,))
    reminders = cursor.fetchall()
    if reminders:
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        for reminder in reminders:
            markup.row(f"Удалить {reminder[0]}")
        markup.row('Назад', 'На главный экран')
        bot.send_message(message.chat.id, "Выберите напоминание для удале-
ния:", reply_markup=markup)
        bot.register_next_step_handler(message, confirm_delete_reminder)
    else:
        bot.send_message(message.chat.id, "У вас нет напоминаний.", re-
ply_markup=health_menu())
# Функция для подтверждения удаления напоминания
def confirm_delete_reminder(message):
    if message.text.startswith("Удалить"):
        reminder_id = int(message.text.split()[1])
        cursor.execute("SELECT text FROM reminders WHERE id = ? AND user_id
= ?", (reminder_id, message.chat.id))
        reminder = cursor.fetchone()
        if reminder:
```

```
            markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
            markup.row('Да', 'Нет')
            bot.send_message(message.chat.id, f"Вы уверены, что хотите уда-
лить напоминание: '{reminder[0]}'?",
                             reply_markup=markup)
            bot.register_next_step_handler(message, lambda m: perform_de-
lete_reminder(m, reminder_id))
        else:
            bot.send_message(message.chat.id, "Напоминание не найдено.",
reply_markup=health_menu())
    elif message.text == 'Назад':
        handle_reminders(message)
    elif message.text == 'На главный экран':
        send_welcome(message)
    else:
        bot.send_message(message.chat.id, "Пожалуйста, выберите одну из оп-
ций.", reply_markup=health_menu())
# Функция для удаления напоминания из базы данных
def perform_delete_reminder(message, reminder_id):
    if message.text == 'Да':
        cursor.execute("DELETE FROM reminders WHERE id = ? AND user_id =
?", (reminder_id, message.chat.id))
        conn.commit()
        bot.send_message(message.chat.id, "Напоминание удалено.", re-
ply_markup=health_menu())
    elif message.text == 'Нет':
        bot.send_message(message.chat.id, "Удаление отменено.", re-
ply_markup=health_menu())
    handle_reminders(message)
# Функция для проверки напоминаний и отправки сообщений
def check_reminders():
    while True:
        cursor.execute("SELECT * FROM reminders")
        reminders = cursor.fetchall()
        current_time = datetime.now(pytz.timezone("UTC"))
        for reminder in reminders:
            user_time_zone = reminder[4]
            if user_time_zone:
                user_current_time = current_time.astimezone(pytz.time-
zone(user_time_zone))
                if user_current_time.strftime("%H:%M") == reminder[3]:
                    bot.send_message(reminder[1], f"Напоминание: {re-
minder[2]}")
                    cursor.execute("DELETE FROM reminders WHERE id = ?",
(reminder[0],))
                    conn.commit()
            time.sleep(60)
# Обработчики кнопок основного меню
@bot.message_handler(func = lambda message: message.text in ["Карта боль-
ниц", "Первая помощь универсальный алгоритм",
                                                             "Поликлиниче-
ская помощь", "Первая помощь", "Здоровье",
                                                             "Связаться с
тех.поддержкой", "Инструкция по работе с ботом",
                                                             "Связаться с
больницей"])
def handle_main_menu(message):
    if message.text == "Карта больниц":
        bot.send_message(message.chat.id,
                         "Вот ссылка на карты ближайших больниц:
https://www.google.com/maps/search/hospital")
    elif message.text == "Первая помощь универсальный алгоритм":
```

```
        send_first_aid_algorithm(message)
    elif message.text == "Поликлиническая помощь":
        send_clinic_help_menu(message)
    elif message.text == "Первая помощь":
        bot.send_message(message.chat.id, "Выберите опцию:", re-
ply_markup=first_aid_menu())
    elif message.text == "Здоровье":
        bot.send_message(message.chat.id, "Выберите опцию:", re-
ply_markup=health_menu())
    elif message.text == "Связаться с тех.поддержкой":
        bot.send_message(message.chat.id,
                         "Вы можете связаться с тех.поддержкой, отправив
сообщение на: \n1. mantrov-i@yandex-team.ru.\n2. @MIA_ivan.")
    elif message.text == "Инструкция по работе с ботом":
        bot.send_message(message.chat.id,
                         "\n1. Выберите нужную опцию из главного меню.\n2.
Следуйте инструкциям бота для ввода необходимых данных.\n3. Используйте
кнопки для навигации по разделам. \n4. Если бот не реагирует на команды,
попробуйте перезапустить бота командой /start.\n5. Если вам нужна помощь,
выберите 'Связаться с тех.поддержкой'.")
    elif message.text == "Связаться с больницей":
        bot.send_message(message.chat.id, "Вы можете связаться с больницей
по следующему аккаунту Telegram: @hospital3test")
# Меню "Первая помощь"
def first_aid_menu():
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Наружное кровотечение', 'Остановка дыхания')
    markup.row('Инородное тело в дыхательных путях', 'Травмы')
    markup.row('Ожоги', 'Обморожения')
    markup.row('Отравления', 'Транспортировка пострадавшего')
    markup.row('Назад')
    return markup
# Обработчик кнопок меню "Здоровье"
@bot.message_handler(func=lambda message: message.text in ["Напоминания",
"Погода", "Рассчитать ИМТ", "Назад"])
def handle_health_menu(message):
    if message.text == "Напоминания":
        handle_reminders(message)
    elif message.text == "Погода":
        msg = bot.send_message(message.chat.id, "Введите название вашего
города для получения текущей погоды:")
        bot.register_next_step_handler(msg, send_weather_info)
    elif message.text == "Рассчитать ИМТ":
        calculate_bmi(message)
    elif message.text == "Назад":
        send_welcome(message)

# Обработчик кнопок меню "Первая помощь"
@bot.message_handler(func=lambda message: message.text in ["Наружное крово-
течение", "Остановка дыхания",
                                                           "Инородное тело
в дыхательных путях", "Травмы", "Ожоги",
                                                           "Обморожения",
"Отравления", "Транспортировка пострадавшего",
                                                           "Назад"])
def handle_first_aid_menu(message):
    if message.text == "Назад":
        send_welcome(message)
    else:
        send_first_aid_image(message.text, message.chat.id)
# Функция для отправки изображений первой помощи
def send_first_aid_image(condition, chat_id):
```

```
    images = {
        "Наружное кровотечение": r"Наружные_кровотечения.png",
        "Остановка дыхания": r"Остановка.png",
        "Инородное тело в дыхательных путях": r"Инородное_тело.png",
        "Травмы": r"Травмы.png",
        "Ожоги": r"Ожоги.png",
        "Обморожения": r"Переохлождение.png",
        "Отравления": r"Отравление.png",
        "Транспортировка пострадавшего": r"Транспорт.png"
    }
    image_path = images.get(condition)
    if image_path:
        with Image.open(image_path) as image:
            image.thumbnail((1024, 1024), Image.LANCZOS)  # Уменьшаем раз-
мер изображения
            temp_image_path = 'temp_image.jpg'
            image.save(temp_image_path, format='JPEG')
            with open(temp_image_path, 'rb') as temp_image:
                bot.send_photo(chat_id, temp_image)
    else:
        bot.send_message(chat_id, "Изображение не найдено.")
# Функция для получения информации о погоде
def send_weather_info(message):
    city = message.text
    api_key = 'TOKEN_API'
    url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&ap-
pid={api_key}&units=metric&lang=ru'
    response = requests.get(url)
    data = response.json()
    if data['cod'] == 200:
        weather_info = f"Погода в {city}:\nТемпература:
{data['main']['temp']}°C\nПогода: {data['weather'][0]['description']}"
        bot.send_message(message.chat.id, weather_info)
    else:
        bot.send_message(message.chat.id, "Город не найден. Попробуйте
снова.")
# Обработчик для расчета ИМТ
@bot.message_handler(func=lambda message: message.text == "Рассчитать ИМТ")
def calculate_bmi(message):
    msg = bot.send_message(message.chat.id, "Введите ваш рост в сантимет-
рах:", reply_markup=cancel_markup())
    bot.register_next_step_handler(msg, process_height_step)
# Функция для получения роста пользователя
def process_height_step(message):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Расчет ИМТ отменен.", re-
ply_markup=main_menu())
        return
    try:
        height = float(message.text) / 100  # Конвертация роста в метры
        msg = bot.send_message(message.chat.id, "Введите ваш вес в кило-
граммах:", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, process_weight_step, height)
    except ValueError:
        msg = bot.send_message(message.chat.id, "Пожалуйста, введите кор-
ректный рост в сантиметрах:", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, process_height_step)
# Функция для получения веса пользователя и расчета ИМТ
def process_weight_step(message, height):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Расчет ИМТ отменен.", re-
ply_markup=main_menu())
```

```
        return
    try:
        weight = float(message.text)
        bmi = weight / (height ** 2)
        bmi = round(bmi, 2)
        bot.send_message(message.chat.id, f"Ваш ИМТ: {bmi}")
        if bmi < 18.5:
            bot.send_message(message.chat.id, "У вас недостаточный вес.")
        elif 18.5 <= bmi < 24.9:
            bot.send_message(message.chat.id, "У вас нормальный вес.")
        elif 25 <= bmi < 29.9:
            bot.send_message(message.chat.id, "У вас избыточный вес.")
        else:
            bot.send_message(message.chat.id, "У вас ожирение.")
    except ValueError:
        msg = bot.send_message(message.chat.id, "Пожалуйста, введите кор-
ректный вес в килограммах:", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, process_weight_step, height)
# Список шагов алгоритма первой помощи
steps = [
    "1. Оценка обстановки и устранение угрожающих факторов",
    "2. Оценка сознания",
    "3. Проверка дыхания и сердцебиения",
    "4. Вызов СМП",
    "5. Проведение СЛР:",
    "5.1 Положить на спину, запрокинуть голову",
    "5.2 Провести 'непрямой массаж сердца'",
    "5.3 Сделать искусственное дыхание",
    "6. Поддержка проходимости дыхательных путей",
    "7. Осмотр пострадавшего",
    "8. Контроль состояния"]
# Функция для отправки алгоритма первой помощи
def send_first_aid_algorithm(message, step=0):
    markup = types.InlineKeyboardMarkup()
    if step < len(steps) - 1:
        next_button = types.InlineKeyboardButton("Дальше",
callback_data=f'next_{step + 1}')
        markup.add(next_button)
    if step > 0:
        back_button = types.InlineKeyboardButton("Назад",
callback_data=f'back_{step - 1}')
        markup.add(back_button)
    home_button = types.InlineKeyboardButton("На главный экран",
callback_data='back_to_menu')
    markup.add(home_button)
    bot.send_message(message.chat.id, steps[step], reply_markup=markup)
# Обработчики для кнопок "Дальше" и "Назад" в алгоритме первой помощи
@bot.callback_query_handler(func=lambda call: call.data.starts-
with('next_'))
def callback_next_step(call):
    step = int(call.data.split('_')[1])
    bot.edit_message_text(chat_id=call.message.chat.id, mes-
sage_id=call.message.message_id, text=steps[step],
                          reply_markup=None)
    send_first_aid_algorithm(call.message, step)
@bot.callback_query_handler(func=lambda call: call.data.starts-
with('back_'))
def callback_back_step(call):
    step = int(call.data.split('_')[1])
    bot.edit_message_text(chat_id=call.message.chat.id, mes-
sage_id=call.message.message_id, text=steps[step],
                          reply_markup=None)
```

```
    send_first_aid_algorithm(call.message, step)
@bot.callback_query_handler(func=lambda call: call.data == 'back_to_menu')
def callback_back_to_menu(call):
    bot.send_message(call.message.chat.id, "Выберите опцию:", re-
ply_markup=main_menu())
# Функция для отправки меню "Поликлиническая помощь"
def send_clinic_help_menu(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Запись на прием к врачу', 'Заболевания')
    markup.row('На главный экран')
    bot.send_message(message.chat.id, "Выберите опцию:", re-
ply_markup=markup)
# Обработчик для записи на прием к врачу
@bot.message_handler(func=lambda message: message.text == "Запись на прием
к врачу")
def handle_appointment(message):
    consent_message = ("Для продолжения записи на прием к врачу, пожалуй-
ста, дайте согласие на сбор и обработку "
                       "персональных данных. Мы будем собирать ваш ID для
организации записей. "
                       "Нажимая 'Согласен', вы даете свое согласие на обра-
ботку ваших данных.")
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True, one_time_key-
board=True)
    markup.add('Согласен', 'Не согласен')
    msg = bot.send_message(message.chat.id, consent_message, re-
ply_markup=markup)
    bot.register_next_step_handler(msg, get_consent)
def get_consent(message):
    if message.text == 'Согласен':
        user_id = message.chat.id
        username = message.from_user.username if message.from_user.username
else 'None'
        save_consent(user_id, username)
        start_appointment_process(message)
    elif message.text == 'Не согласен':
        bot.send_message(message.chat.id, "К сожалению, без вашего согласия
мы не можем продолжать работу.", reply_markup=types.ReplyKeyboardRemove())
    else:
        consent_message = ("Для продолжения записи на прием к врачу, пожа-
луйста, дайте согласие на сбор и обработку "
                           "персональных данных. Мы будем собирать ваш ID
для организации записей. "
                           "Нажимая 'Согласен', вы даете свое согласие на
обработку ваших данных.")
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True,
one_time_keyboard=True)
        markup.add('Согласен', 'Не согласен')
        msg = bot.send_message(message.chat.id, consent_message, re-
ply_markup=markup)
        bot.register_next_step_handler(msg, get_consent)
def save_consent(user_id, username):
    cursor.execute("INSERT OR REPLACE INTO users (user_id, consent,
username) VALUES (?, ?, ?)", (user_id, 1, username))
    conn.commit()
def start_appointment_process(message):
    msg = bot.send_message(message.chat.id, "Введите ваши ФИО:", re-
ply_markup=cancel_markup())
    bot.register_next_step_handler(msg, process_name_step)
# Функция для получения ФИО пользователя
def process_name_step(message):
    if message.text == 'Отмена':
```

```
        bot.send_message(message.chat.id, "Запись на прием отменена.", re-
ply_markup=main_menu())
        return
    try:
        name = message.text
        if re.match(r'^[А-ЯЁ][а-яё]+\s[А-ЯЁ][а-яё]+\s[А-ЯЁ][а-яё]+$',
name):
            msg = bot.send_message(message.chat.id, "Введите ваш возраст:",
reply_markup=cancel_markup())
            bot.register_next_step_handler(msg, process_age_step, name)
        else:
            msg = bot.send_message(message.chat.id,
                                   "Неверный формат. Введите ваши ФИО в
формате 'Фамилия Имя Отчество':", reply_markup=cancel_markup())
            bot.register_next_step_handler(msg, process_name_step)
    except Exception as e:
        bot.reply_to(message, 'Ошибка. Попробуйте снова.', re-
ply_markup=main_menu())
# Функция для получения возраста пользователя
def process_age_step(message, name):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Запись на прием отменена.", re-
ply_markup=main_menu())
        return
    try:
        age = int(message.text)
        if 0 < age < 120:
            msg = bot.send_message(message.chat.id, "Введите ваш пол
(М/Ж):", reply_markup=cancel_markup())
            bot.register_next_step_handler(msg, process_gender_step, name,
age)
        else:
            msg = bot.send_message(message.chat.id, "Неверный возраст. Вве-
дите ваш возраст (число от 1 до 120):", reply_markup=cancel_markup())
            bot.register_next_step_handler(msg, process_age_step, name)
    except ValueError:
        msg = bot.send_message(message.chat.id, "Возраст должен быть чис-
лом. Введите ваш возраст:", reply_markup=cancel_markup())
        bot.register_next_step_handler(msg, process_age_step, name)
    except Exception as e:
        bot.reply_to(message, 'Ошибка. Попробуйте снова.', re-
ply_markup=main_menu())
# Функция для получения пола пользователя
def process_gender_step(message, name, age):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Запись на прием отменена.", re-
ply_markup=main_menu())
        return
    try:
        gender = message.text.upper()
        if gender in ['М', 'Ж']:
            msg = bot.send_message(message.chat.id, "Введите ваши симптомы
(например, головная боль, кашель, жар):", reply_markup=cancel_markup())
            bot.register_next_step_handler(msg, process_condition_step,
name, age, gender)
        else:
            msg = bot.send_message(message.chat.id, "Неверный формат. Вве-
дите ваш пол (М/Ж):", reply_markup=cancel_markup())
            bot.register_next_step_handler(msg, process_gender_step, name,
age)
    except Exception as e:
        bot.reply_to(message, f'Ошибка: {e}', reply_markup=main_menu())
```

```
def process_condition_step(message, name, age, gender):
    if message.text == 'Отмена':
        bot.send_message(message.chat.id, "Запись на прием отменена.", re-
ply_markup=main_menu())
        return
    try:
        condition = message.text  # Здесь пользователи могут ввести любые
симптомы
        user_id = message.chat.id
        cursor.execute("INSERT INTO appointments (user_id, name, age, gen-
der, condition) VALUES (?, ?, ?, ?, ?)",
                       (user_id, name, age, gender, condition))
        conn.commit()
        bot.send_message(message.chat.id, "Вы успешно записаны на прием!",
reply_markup=main_menu())
    except Exception as e:
        bot.reply_to(message, f'Ошибка: {e}', reply_markup=main_menu())
# Обработчик для выбора заболевания
@bot.message_handler(func=lambda message: message.text == "Заболевания")
def handle_diseases(message):
    cursor.execute("SELECT name FROM diseases")
    diseases = [row[0] for row in cursor.fetchall()]
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    for disease in diseases:
        markup.row(disease)
    markup.row('Назад', 'На главный экран')
    bot.send_message(message.chat.id, "Выберите заболевание:", re-
ply_markup=markup)
# Обработчик выбора конкретного заболевания
@bot.message_handler(func=lambda message: True)
def handle_disease_info(message):
    cursor.execute("SELECT symptoms, diagnosis, treatment FROM diseases
WHERE name = ?", (message.text,))
    disease_info = cursor.fetchone()
    if disease_info:
        symptoms, diagnosis, treatment = disease_info
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        markup.row('Симптомы', 'Диагностика', 'Лечение')
        markup.row('Назад', 'На главный экран')
        bot.send_message(message.chat.id, f"Выберите информацию по
{message.text}:", reply_markup=markup)
        bot.register_next_step_handler(message, lambda msg: send_dis-
ease_info(msg, message.text, symptoms, diagnosis, treatment))
    else:
        bot.send_message(message.chat.id, "Пожалуйста, выберите одну из оп-
ций.", reply_markup=main_menu())
def send_disease_info(message, disease, symptoms, diagnosis, treatment):
    if message.text == 'Симптомы':
        info = symptoms
    elif message.text == 'Диагностика':
        info = diagnosis
    elif message.text == 'Лечение':
        info = treatment
    else:
        handle_disease_info(message)
        return
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    markup.row('Назад', 'На главный экран')
    bot.send_message(message.chat.id, info, reply_markup=markup)
    bot.register_next_step_handler(message, lambda msg: handle_dis-
ease_info_return(msg, disease))
```

```
def handle_disease_info_return(message, disease):
    cursor.execute("SELECT symptoms, diagnosis, treatment FROM diseases
WHERE name = ?", (disease,))
    disease_info = cursor.fetchone()
    if disease_info:
        symptoms, diagnosis, treatment = disease_info
        markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
        markup.row('Симптомы', 'Диагностика', 'Лечение')
        markup.row('Назад', 'На главный экран')
        bot.send_message(message.chat.id, f"Выберите информацию по
{disease}:", reply_markup=markup)
        bot.register_next_step_handler(message, lambda msg: send_dis-
ease_info(msg, disease, symptoms, diagnosis, treatment))
# Запуск бота с обработкой ошибок и переподключением
def run_bot():
    while True:
        try:
            bot.polling(none_stop=True, interval=0, timeout=20)
        except Exception as e:
            print(f"Ошибка: {e}")
            time.sleep(15)
# Запуск функции проверки напоминаний в отдельном потоке
def run_reminder_check():
    while True:
        check_reminders()
        time.sleep(60)
if __name__ == "__main__":
    threading.Thread(target=run_reminder_check).start()
    run_bot()
```

Приложение В. Диаграмма вариантов использования чат-бота

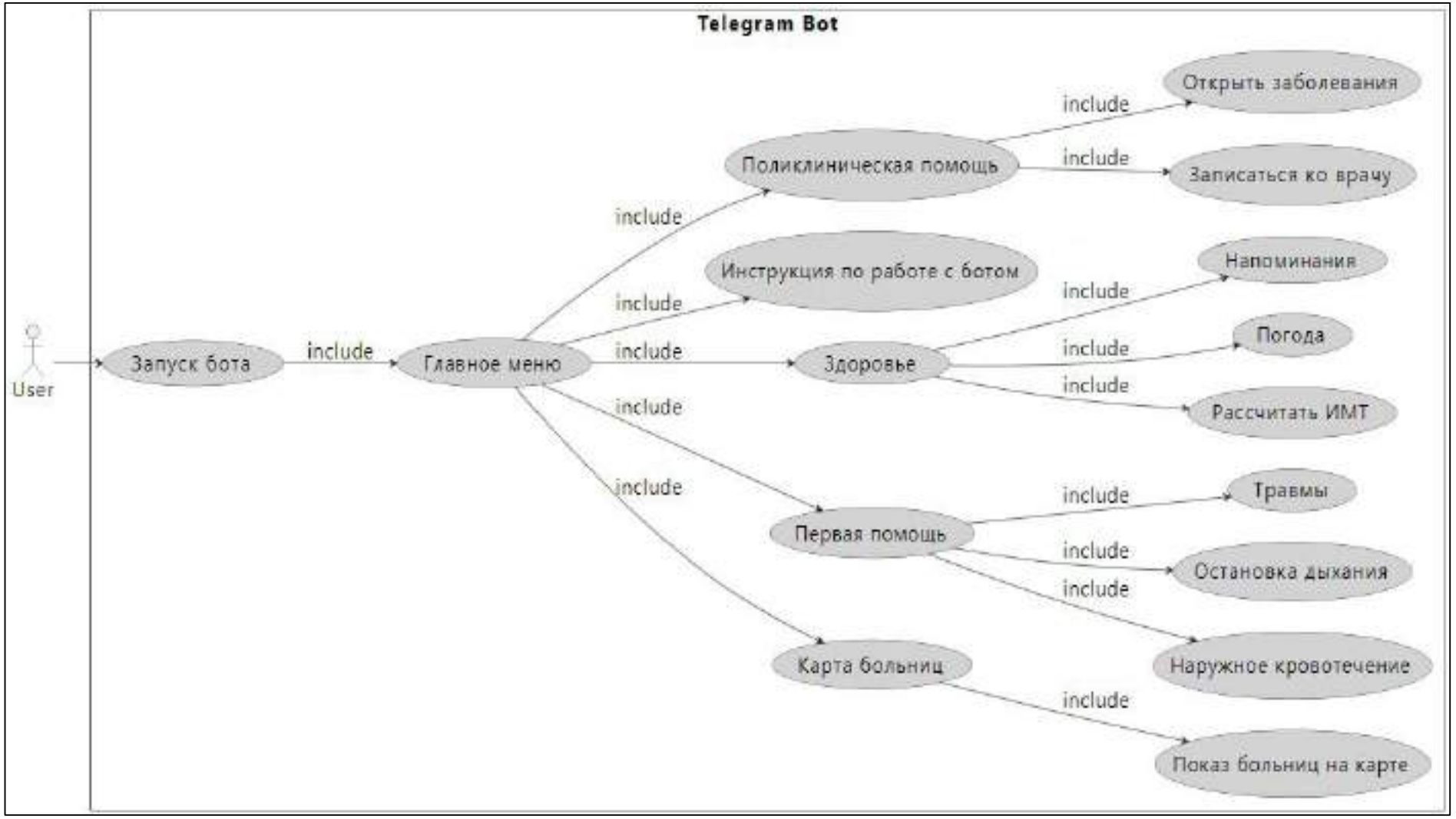

Рисунок 1 – Диаграмма вариантов использования чат-бота