СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, который отмечается крайним бурным ростом информационных технологий и социальных платформ, взаимодействие между людьми становится всё более необходимым. Одним из распространённых способов, который значительно упростил общение между людьми, является мессенджер телеграмм. Данный мессенджер предоставляет пользователям не только возможность вести общение через отправку сообщений и совершения звонков, но и предоставляет широкий функционал, включая возможность создания чат ботов для различных задач. Чат боты, созданные в мессенджере телеграмм, набирают популярность за счёт своей простоты и эффективности, с их помощью можно осуществлять онлайн заказы с интернет магазинах, производить запись на приём, получать свежие новости, производить интеграцию с другими сервисами и т.д. Их использование позволяет значительно сэкономить время и упростить взаимодействие с различными службами и сервисами.

Актуальность темы обусловлена тем, что в условиях глобализации и цифровизации тяжело обойтись без каких-либо связей, так как при их отсутствии в жизни могут возникнуть проблемы, которые крайне тяжело решить без чьей-либо помощи, чем и обусловлена существующая потребность в расширении круга общения и формирования новых социальных связей. В расширении круга общения может помочь телеграмм бот так как это достаточно удобный инструмент, который можно настроить для данной цели. Разработка телеграмм бота для поиска друзей имеет актуальность в настоящее время, в связи с необходимостью расширения связей в современном мире. За последние годы интерес к мессенджеру телеграмм среди людей возрос, так же на данный рост повлияла пандемия COVID-19 2020 года которая вынудила людей использовать дистанционный метод общения. Стоит отметить, что на платформе телеграмм существует широкий спектр ботов, предлагающие различные услуги, от обучения до развлечений. Однако среди них не хватает специализированных решений для поиска друзей и расширения социальных связей,

что подчёркивает уникальность и востребованность разработки данного телеграмм бота.

Целью данной работы является разработка функционального телеграмм бота, способного принимать введённые данные пользователем, и на основе полученных данных предлагать пользователю актуальные совпадения. Для достижения цели, поставленной в работе, были определены следующие задачи:

1) Анализ проблемы;
2) Анализ существующих решений;
3) Изучить принцип работы телеграмм бота с помощью интернет ресурсов;
4) Составить план по разработке телеграмм бота;
5) Разработка телеграмм бота с функционалом по обработке запросов пользователей и внедрение алгоритмов, позволяющих осуществлять поиск на основе общих интересов;
6) Провести тестирование на работоспособность;
7) Провести расчёт экономической эффективности.

Объектом работы выступает мессенджер телеграмм как платформа для создания и использования ботов, а также взаимодействие пользователей в рамках этой платформы. Предметом работы является процесс разработки телеграмм бота, предназначенного для поиска друзей.

Выпускная квалификационная работа состоит из следующих разделов:

- Введение: Обоснование актуальности, формулировка цели, задач, объекта и предмета исследования.
- Теоретическая часть: Анализ предприятия, анализ предметной области, Теоретические основы разработки телеграмм бота, постановка задачи, разработка технического задания.
- Практическая часть: План по разработке телеграмм бота, проектирование логики.
- Заключение: Ключевые результаты исследования и предложения по внедрению бота.
- Приложения: Исходный код.

ГЛАВА 1 Теоретические основы разработки телеграмм бота

1.1 Анализ предприятия

Полное наименование: АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «БАЛТИЙСКИЙ ИНФОРМАЦИОННЫЙ ТЕХНИКУМ». Сокращенное название: АНО ПО «БИТ». Дата создания образовательной организации: 28 августа 1999 года. Место нахождения образовательной организации: 236016, Российская Федерация, г. Калининград, ул. Литовский вал, д. 38, литер А1, пом. YII, подъезд 6. Учредитель: Сергеев Вячеслав Владимирович.

«Балтийский информационный техникум» создан в 1999 году. С тех пор было выпущено более 2000 квалифицированных специалистов в области информационных технологий, востребованных на региональном рынке труда.

Управление Техникумом осуществляется в соответствии с законодательством Российской Федерации, Калининградской области с учетом особенностей, установленных Федеральным законом «Об образовании в Российской Федерации» и Уставом Техникума. Организационная структура представлена на рис. 1.1.

Наименование структурного подразделения	Место нахождения структурного подразделения	Должность руководителя структурного подразделения	Ф.И.О. руководителя структурного подразделения	Положения о структурных подразделениях
Учебный отдел	Кабинет 3-23	Заведующая отделом по учебной работе	Дорофеева Екатерина Евгеньевна	Положение об учебном отделе
Отделение по воспитательной работе	Кабинет 3-23	Заведующая отделением по воспитательной работе	Небиева Надежда Игоревна	Положение об отделении воспитательной работы
Отдел содействия занятости студентов и трудоустройству выпускников	Кабинет 3-16	Руководитель	Трусова Софья Андреевна	Положение об отделе занятости студентов и трудоустройству выпускников
Учебно-производственный комплекс (УПК)	Кабинет 3-18	Руководитель	Трусова Софья Андреевна	Положение об учебно-производственном комплексе

Рис. 1.1 Структура образовательной организации

Органами управления Техникума являются:

- Учредитель, директор техникума – Сергеев Вячеслав Владимирович, который осуществляет непосредственное управление деятельностью Техникума.

К исполнительным коллегиальным органам управления относятся:

- Общее собрание (конференция) работников и обучающихся Техникума: постоянно действующий коллегиальный орган управления, который составляют граждане, участвующие своим трудом в деятельности Техникума и представители обучающихся.
- Общее собрание (конференция) работников и обучающихся собирается по мере необходимости, но не реже раза в год.
- Педагогический совет Техникума: в целях развития и совершенствования учебно-воспитательного процесса, повышения профессионального мастерства и творческого роста педагогических работников в Техникуме действует постоянно действующий коллегиальный орган управления – Педагогический совет, объединяющий педагогических работников Техникума. В состав Педагогического совета входят: директор, его заместители, педагогические работники.
- Совет Техникума: постоянно действующий коллегиальный орган управления, формируемый посредством процедур выборов. Совет Техникума избирается для осуществления общего руководства Техникумом. В состав Совета Техникума могут входить представители заказчиков, спонсоров, местных органов власти и местного самоуправления, работодателей.

1.2 Анализ предметной области

В настоящее время в обществе существует проблема с социализацией, многим людям тяжело найти себе собеседника в силу различных интересов. Несмотря на наличие восьми миллиардов людей, найти собеседника до сих пор остаётся не простой задачей, из-за разных взглядов на жизнь, политики, увлечений, а также из-за разного типа личностей. По состоянию на 2023 год некоторые исследования говорят о значительных проблемах одиночества в России. Одиночество среди молодежи, исследование, проведенное в 2021 году фондом “Общественное мнение”, показало,

что 27% граждан России в возрасте от 18 до 25 лет ощущают себя одинокими. Согласно опросу в России, проведенному “Левада-центром” в 2021 году, около 30% россиян заявили, что испытывают чувство одиночества. Отсутствие общения и социальных взаимодействий может оказывать серьёзное воздействие на психическое здоровье человека.

Основные аспекты, и как это может проявляться:

- Увеличение чувства одиночества. При недостаточном общении люди часто начинают чувствовать себя одинокими, что может привести к депрессии и тревожным расстройствам;
- Снижение самооценки. Недостаток общения может повредить самооценку и уверенность в себе, поскольку человек воспринимает себя как ненужного или непривлекательного для других;
- Увеличение стресса. Социальная изоляция может увеличить уровень стресса, усиливая негативные чувства и эмоции, такие как тревога и печаль.

В психологии выделяют два типа личностей: экстраверт и интроверт. Экстраверт – это тип личности, который характеризуется большим стремлением к внешнему миру, общению с людьми и социальной активности. Интроверты – это тип личности противоположный экстраверту, интроверты - это люди которым комфортно проводить время наедине, в своих мыслях или в узком кругу близких людей.

Если человек с типом лично экстраверт поставит себе задачу расширить круг общения, то данную задачу он сможет выполнить без каких-либо трудностей, так как экстравертам крайне легко ужиться в социуме. Но если человек с типом личности интроверт поставит себе задачу по расширению круга общения, то данную задачу с большой вероятностью выполнить не сможет, так как в силу своей личности, данная задача будет казаться крайне тяжелой. Поэтому, для расширения круга общения, люди с типом личности интроверт будут надеяться на всемирную паутину интернет.

В условиях, когда традиционные формы общения могут быть затруднены, онлайн-дружба, особенно в контексте кооперативных игр, представляет собой важную альтернативу. Участие в таких играх создает уникальные возможности для формирования новых знакомств и укрепления социальных связей.

Кооперативные игры требуют взаимодействия и совместной работы, что способствует созданию атмосферы доверия и взаимопонимания. Игроки, объединенные общими интересами и целями, могут легко находить общий язык, что особенно важно для интровертов, которые могут испытывать трудности в общении в реальной жизни. Виртуальная среда позволяет им взаимодействовать с другими людьми, не испытывая при этом давления, связанного с личными встречами. Кроме того, общение в рамках кооперативных игр может значительно улучшить психическое здоровье участников. Исследования показывают, что совместные действия и взаимодействия в играх помогают снизить уровень стресса и тревожности, а также способствуют улучшению настроения. Игроки могут делиться своими переживаниями, поддерживать друг друга и находить эмоциональную поддержку в сложные времена, что особенно важно в условиях социальной изоляции.

Общие интересы, такие как увлечение играми, становятся основой для успешных знакомств и взаимодействий. Люди, которые разделяют страсть к определенным играм, могут легче находить общий язык и строить отношения, основанные на взаимопонимании и уважении. Это создает более комфортную среду для общения, где каждый может быть собой и не бояться осуждения.

Таким образом, онлайн-дружба в контексте кооперативных игр не только помогает людям расширять круг общения, но и служит важным инструментом для борьбы с одиночеством и социальной изоляцией. Она открывает новые горизонты для взаимодействия и позволяет людям находить поддержку и понимание в виртуальном пространстве, что особенно актуально в современном мире.

В распространенные инструменты для расширения круга общения с помощью всемирной сети интернет входят: социальные сети, форумы, игровые платформы, видеочаты. На данный момент существуют различные площадки для расширения круга общений, в пример можно привести несколько сайтов, такие как Coop-Land, Ansedo и GameTree. Эти площадки отмечаются хорошим и интуитивно понятным дизайном, а также большим количеством пользователей. Но стоит отметить, что многие платформы не обеспечивают достаточную защиту личных данных пользователей. Например, Coop-Land и Ansedo не всегда предоставляют четкую

информацию о том, как обрабатываются и хранят данные пользователей. К тому же, у данных сайтов есть проблемы с пользователями, которые рекламируют свои площадки, из-за чего, объявления обычных пользователей, которые заинтересованы в поиске друга, теряются среди рекламы. Необходимая регистрация для использования этих площадок также может отпугнуть некоторых пользователей, особенно тех, кто предпочитает анонимность.

Разнообразие платформ для общения также заслуживает внимания. Существуют приложения для знакомств, такие как Tinder и Bumble. Специализированные сообщества по интересам, такие как Reddit и Discord, предоставляют возможность общения с единомышленниками и обмена опытом. Недостаточная модерация на некоторых платформах может приводить к токсичному поведению пользователей, что затрудняет поиск качественного общения. Это подчеркивает важность наличия эффективных инструментов для управления контентом и защиты пользователей от кибербуллинга.

Существуют и альтернативные решения, акцентирующие внимание на безопасности и конфиденциальности, такие как Signal или телеграмм, где пользователи могут общаться в защищенной среде. Эти платформы предлагают шифрование сообщений и дополнительные настройки конфиденциальности, что делает их более привлекательными для тех, кто заботится о своей безопасности.

1.3 Теоретические основы разработки телеграмм бота

В нынешнее время, большей популярностью для общения пользуются мессенджеры, одним из наиболее популярных мессенджеров, который завоевал широкую аудиторию, является телеграмм. Этот кроссплатформенный мессенджер предоставляет своим пользователям возможность обмениваться не только текстовыми сообщениями, но и голосовыми, а также видеосообщениями, что делает его универсальным инструментом для общения. На сегодняшний день телеграмм занимает второе место в рейтинге социальных сетей по версии App Store, что свидетельствует о его значительной популярности и востребованности среди пользователей. В России аудитория телеграмм превышает 50 миллионов человек, что подчеркивает его влияние на коммуникационные процессы в стране.

Одной из ключевых особенностей телеграмм, которая отличает его от других социальных сетей, является возможность создания и использования телеграмм ботов. Телеграмм бот — это специализированный аккаунт в мессенджере, который запрограммирован на автоматическое выполнение различных действий. Эти действия могут включать в себя рассылку рекламных сообщений, предоставление консультаций по товарам и услугам, а также осуществление продаж. Несмотря на наличие уже существующих решений в данной области, которые, как правило, имеют определенные недостатки, было принято решение о разработке собственного проекта. Основой для этого послужил телеграмм бот, который обладает широким функционалом, высоким уровнем безопасности, простотой в использовании, а также требует минимальных финансовых вложений и времени на разработку.

Несмотря на наличие множества уже существующих решений, которые, безусловно, имеют свои достоинства, они также обладают рядом существенных недостатков, которые могут значительно ограничивать их эффективность и удобство использования. В связи с этим, было принято решение о создании собственного проекта, который будет основываться на использовании телеграмм бота. Это решение было выбрано по нескольким важным причинам.

Во-первых, телеграмм боты предлагают широкий спектр функциональных возможностей, что делает их весьма универсальными инструментами для общения и взаимодействия с пользователями. Они могут выполнять различные задачи, начиная от автоматизации процессов и заканчивая предоставлением информации в реальном времени. Это позволяет создать более интерактивный и удобный опыт для пользователей, что, безусловно, является важным аспектом в современном мире, где скорость и качество взаимодействия играют ключевую роль.

Во-вторых, безопасность является одним из главных приоритетов в нашем проекте. Телеграмм известен своим высоким уровнем защиты данных и конфиденциальности, что делает его идеальной платформой для создания бота, который будет обрабатывать личные данные пользователей. В условиях, когда вопросы безопасности и защиты личной информации становятся все более

актуальными, использование телеграмм как основы для нашего проекта позволяет нам гарантировать пользователям надежность и защиту их данных.

В-третьих, простота использования телеграмм ботов является еще одним значительным преимуществом. Пользователи могут легко взаимодействовать с ботом, не требуя от них специальных технических знаний или навыков. Это делает наш проект доступным для широкой аудитории, включая тех, кто может не иметь опыта работы с более сложными платформами или приложениями. Удобный интерфейс и интуитивно понятные команды позволяют пользователям быстро освоиться и начать получать удовольствие от общения.

Кроме того, создание телеграмм бота требует относительно небольших вложений как в денежном, так и в временном плане. Это позволяет нам сосредоточиться на разработке и улучшении функционала, а не тратить ресурсы на сложные технические решения или дорогостоящие платформы. Таким образом, мы можем эффективно использовать наши ресурсы и время для достижения наилучших результатов.

API это программный интерфейс, то есть описание способов взаимодействия одной компьютерной программы с другими. Обычно входит в описание какого-либо интернет-протокола, программного каркаса или стандарта вызовов функций операционной системы. API, или интерфейс программирования приложений, представляет собой набор правил и протоколов, позволяющих различным программным компонентам взаимодействовать друг с другом. Основная цель API заключается в удовлетворении потребностей разработчиков, позволяя им использовать функциональность одной программы в другой, не углубляясь в детали внутренней реализации каждой из них.

API обеспечивает интероперабельность между системами, позволяя им обмениваться данными и выполнять совместные задачи, что способствует созданию более гибкой архитектуры программного обеспечения. Кроме того, API тоже делает приложения более модульными, позволяя разработчикам строить новые приложения, базируясь на существующих сервисах и библиотеках. Более того, современный API предлагает механизмы аутентификации и авторизации, такие как OAuth, что

обеспечивает защиту данных от несанкционированного доступа. API управляет данными, предоставляя механизм для их запроса, получения, обновления и удаления, что становится особенно важным при работе с базами данных. Также следует отметить, что API позволяет добавлять новые функции без изменения существующих систем, делая приложения более адаптивными к изменениям требований бизнеса.

Взаимодействие API с ботами, такими как чат-боты, происходит следующим образом. Бот отправляет запрос к определенному API, используя предопределенные методы HTTP, такие как GET, POST или PUT. Эти запросы могут содержать параметры, определяющие, какую информацию требуется получить или изменить. Сервер, принимающий этот запрос, взаимодействует с базой данных или другими сервисами для выполнения запрашиваемой операции. Например, в случае чат-бота это может быть поиск информации о продукте или проверка статуса заказа. После обработки запроса сервер возвращает ответ, обычно в формате JSON или XML, который бот интерпретирует для предоставления пользователю запрашиваемой информации или выполнения действия. Таким образом, API играет ключевую роль в расширении возможностей ботов, позволяя им интегрироваться с внешними системами и предоставлять пользователям актуальную и полезную информацию.

Взаимодействие бота с пользователями и сервером телеграмм представляет собой сложный процесс, основанный на использовании API телеграмм, который обеспечивает обмен данными между клиентом и сервером. Боты в телеграмм функционируют как автоматизированные программы, способные обрабатывать сообщения и выполнять команды, предоставляя пользователям различные услуги и информацию.

Когда пользователь отправляет сообщение боту, это сообщение передается на сервер телеграмм. Сервер обрабатывает входящие данные и направляет их на соответствующий бот, используя уникальный токен, который идентифицирует конкретного бота. Бот, в свою очередь, получает это сообщение через API телеграмм, используя методы, такие как getUpdates или Webhook. Метод getUpdates позволяет боту периодически запрашивать сервер о новых сообщениях, тогда как Webhook

обеспечивает более эффективный подход, при котором сервер телеграмм отправляет уведомления о новых сообщениях непосредственно на указанный URL-адрес бота.

После получения сообщения бот анализирует его содержимое, определяет, какие действия необходимо предпринять, и формирует ответ. Этот ответ может включать текстовые сообщения, изображения, кнопки и другие элементы интерфейса. Бот отправляет ответ обратно на сервер телеграмм, используя методы API, такие как sendMessage или sendPhoto. Сервер телеграмм затем обрабатывает этот ответ и передает его пользователю. Таким образом, взаимодействие бота с пользователями и сервером телеграмм осуществляется через четко определенные протоколы и методы API, что позволяет обеспечить эффективный обмен данными и высокую степень автоматизации процессов.

Телеграмм боты пишутся на высокоуровневом языке программирования Python, с использованием библиотеки telebot. Эта библиотека, основанная на телеграмм Bot API, предоставляет разработчикам интуитивно понятный интерфейс, который значительно упрощает процесс создания и настройки ботов.

Одним из ключевых аспектов, способствующих популярности telebot, является её простота в использовании. Разработчики, даже не обладающие глубокими знаниями в области программирования, могут быстро освоить основные принципы работы с библиотекой и начать создавать функциональные боты. Это достигается благодаря лаконичному синтаксису и хорошо структурированной документации, которая предоставляет исчерпывающие примеры и объяснения. Библиотека поддерживает все основные функции телеграмм API, что позволяет разработчикам реализовывать широкий спектр возможностей, таких как отправка и получение сообщений, обработка команд, работа с кнопками и интеграция различных медиаформатов. Это делает библиотеку telebot универсальным инструментом для создания как простых, так и сложных ботов, способных выполнять разнообразные задачи.

Кроме того, библиотека telebot поддерживает асинхронное программирование, что позволяет эффективно обрабатывать множество запросов одновременно. Это

особенно важно для ботов, которые могут сталкиваться с высокой нагрузкой и требуют быстрой реакции на действия пользователей.

1.4 Постановка задачи

В рамках данной работы требуется разработать телеграмм бота с использованием языка программирования Python. Необходимо разработать базу данных, состоящую из нескольких таблиц. Каждая таблица будет хранить определенные данные. Например, потребуется таблица для хранения информации о созданных объявлениях, а также таблица для зарегистрированных пользователей и таблица для пользователей, которые были заблокированы. Для обеспечения регистрации новых пользователей необходимо создать функцию, которая будет добавлять их в базу данных. Также следует реализовать функционал, позволяющий пользователям создавать собственные объявления, включающие возможность добавления изображения, страны, имени, возраста, интересуемой игры и дополнительного текста, в котором можно указать критерии отбора. Кроме того, необходимо предусмотреть функцию для просмотра собственных объявлений, а также возможность просмотра чужих объявлений по таким критериям, как возраст, игра и страна. Также необходимо реализовать отдельный функционал для модераторов, которые будут проверять новые объявления перед их публикацией в общий доступ, и блокировка нежелательных пользователей.

1.5 Разработка технического задания

1. Наименование: Телеграмм бота на Python для поиска друзей, ориентированного на студентов техникума и других пользователей.
2. Целевая аудитория: Основной целевой аудиторией бота являются студенты техникума, однако бот будет общедоступным, что позволяет использовать его и другим пользователям.
3. Функции бота.

   3.1. Функции для пользователей:

   1. Создание объявлений: Пользователи могут создавать объявления о поиске друзей с указанием интересов.

2. Просмотр чужих объявлений: Возможность просматривать объявления других пользователей.
3. Просмотр собственного объявления: Пользователи могут видеть свои собственные объявления.

3.2 Функции для модераторов:

1. Пропуск объявления: Модераторы могут одобрять объявления, которые прошли проверку.
2. Отклонение объявления: Модераторы могут отклонять объявления, которые не соответствуют требованиям.
3. Блокировка пользователя: Возможность блокировки пользователей, создавших неподобающие объявления.

4. Пользовательский интерфейс: Для запуска бота пользователю необходимо ввести команду /start. Дальнейшее взаимодействие будет происходить через отправку сообщений боту и нажатие кнопок, предоставляемых ботом.
5. Структура базы данных.

   База данных будет состоять из пяти таблиц:

   1. ads_table (таблица для объявлений пользователей):
   2. moderation (статус объявления: активно, на рассмотрении, не прошло проверку)
   3. banned_users (таблица для заблокированных пользователей):
   4. id_user_table (таблица для хранения ID пользователей, согласившихся на условия пользования)
   5. moderation_table (таблица для хранения ID пользователей с привилегиями модератора)
6. Алгоритмы поиска.

   Поиск друзей будет осуществляться по следующим критериям:

   - Интересующая игра
   - Минимальный возраст
   - Страна проживания

7. Безопасность и конфиденциальность.

   Использование телеграмм обеспечивает определённый уровень защиты данных пользователей. В условиях пользования ботом будет указано, что разработчик не несёт ответственности за потерю данных.

8. Технологические требования.

   Для разработки бота будут использованы следующие библиотеки:

   - Telebot
   - mysql.connector

9. Сроки разработки.

   Предполагаемый срок выполнения проекта составляет примерно 28 дней.

ГЛАВА 2 Практическая реализация телеграмм бота

2.1 План по разработке телеграмм бота

Перед началом разработки телеграмм бота необходимо составить план действий. В первую очередь следует определить подходящую рабочую среду для разработки. Затем необходимо выбрать систему управления базами данных для эффективного хранения и управления данными. После выбора программного обеспечения можно перейти к разработке основной логики кода. Как только логика будет определена, можно приступить к непосредственной разработке телеграмм бота. В итоге, план разработки телеграмм бота может быть представлен в следующем виде:

- Выбор рабочей среды программного кода;
- Выбор СУБД;
- Придумать основную логику программного кода;
- Разработка телеграмм бота;
- Тестирование телеграмм бота.

2.2 Логика кода

Самой популярной средой разработки для языка программирования Python является Pycharm. PyCharm это интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains. Она предоставляет множество инструментов и функций, которые упрощают процесс разработки. Именно эта среда разработки будет использоваться для дальнейшего написания программного кода.

После выбора среды разработки, необходимо выбрать систему управления базами данных. Одной из популярных систем управления баз данных является MySQL Workbench. MySQL Workbench это инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL. Основные функции MySQL Workbench включают:

- Моделирование данных — возможность создания и редактирования схем баз данных с помощью визуального редактора;

- SQL редактор — инструмент для написания и выполнения SQL-запросов с подсветкой синтаксиса и автозавершением;
- Управление соединениями — возможность подключения к нескольким серверам MySQL и управления ими;
- Администрирование — инструменты для управления пользователями, настройками сервера и мониторинга производительности;
- Импорт и экспорт данных — функции для работы с данными, включая импорт из различных форматов и экспорт в них.

После того как определились с выбором программного обеспечения, можно переходить к разработке основной логики программного кода. Логика программного кода представлена на рис. 2.1.

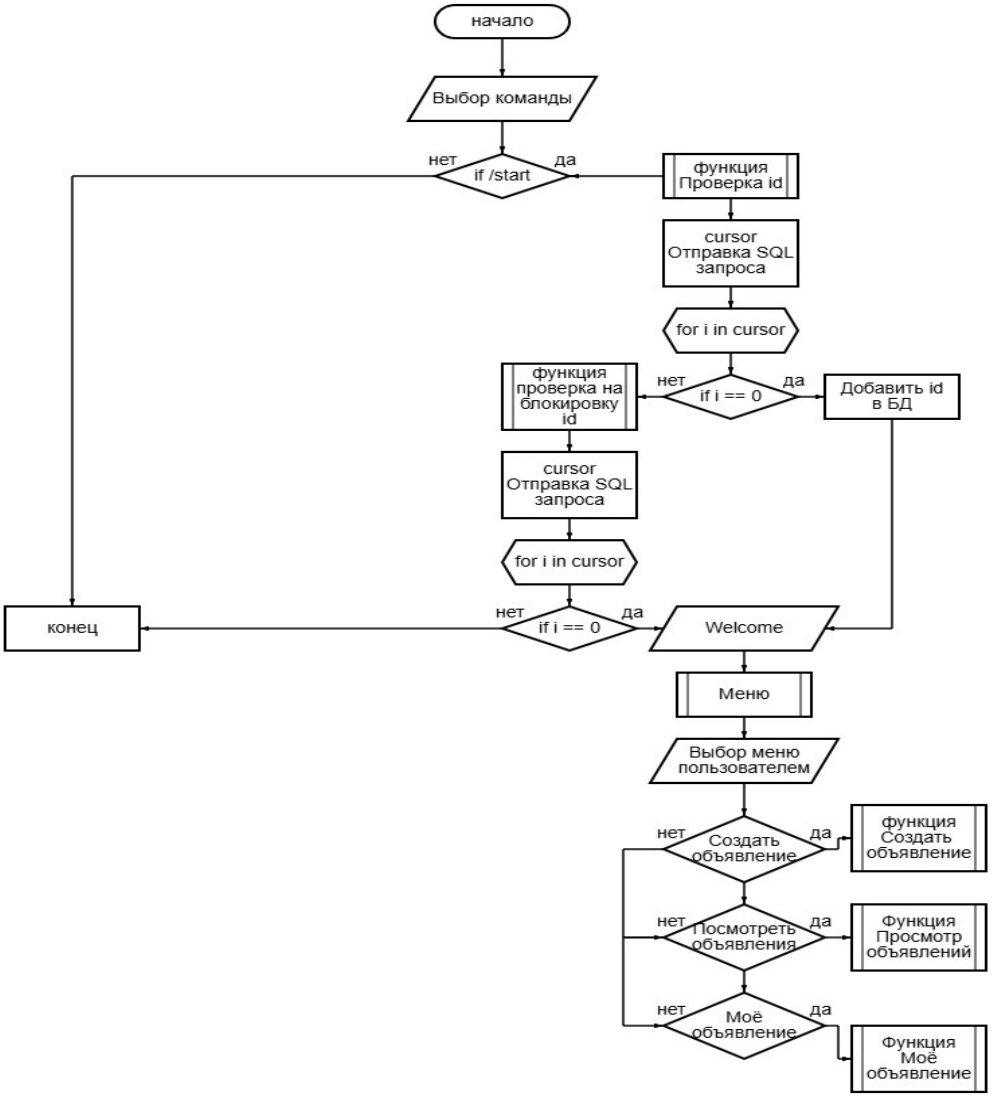

Рис. 2.1 Блок схема логики программного кода

Чтобы начать общение с телеграмм ботом, пользователь должен отправить команду /start в чат, после чего бот отправит SQL запрос в базу данных чтобы проверить, есть ли айди пользователя в базе данных или нет, если его нет, то бот добавит айди пользователя в базу данных и отправит сообщение об отказе ответственности со стороны разработчика, нажав на кнопку “Согласен”, бот отправит приветственное сообщение пользователю предоставив ему меню состоящее из трёх пунктов, таких как “Создать объявление”, “Найти объявления” и “Моё объявление”.

При выборе пункта "Создать объявление" бот инициирует SQL-запрос к базе данных с целью проверки наличия готового объявления, связанного с идентификатором пользователя. В случае обнаружения существующего объявления, бот уведомит пользователя о том, что данное объявление уже имеется. Если же объявления не существует, бот предложит пользователю создать новое. Процесс создания объявления начинается с выбора, будет ли добавлено изображение. В случае выбора опции добавления изображения, пользователь должен отправить боту любое изображение, которое будет включено в объявление. После этого бот запросит у пользователя ввести своё имя. Если пользователь решит не добавлять изображение, бот сразу попросит ввести имя пользователя. После ввода имени бот запросит указать возраст пользователя, который должен быть в пределах от 10 до 60 лет. Далее бот попросит ввести текст объявления, интересующую видеоигру и страну проживания пользователя. По завершении опроса пользователю будет отправлено сообщение, содержащее введённые данные, такие как имя, возраст, страна, описание и игра. Пользователь получит возможность выбрать: отправить объявление на проверку или внести изменения. В случае выбора отправки на проверку, объявлению будет присвоен статус "На рассмотрении".

При выборе пункта "Найти объявления" бот запрашивает у пользователя название видеоигры, затем предлагает указать возраст, до которого будет осуществляться поиск, и, наконец, просит ввести страну, из которой будет производиться поиск. После ввода пользователем запрашиваемых данных, бот инициирует SQL-запрос к базе данных для получения объявлений, соответствующих указанным критериям. В случае нахождения объявлений, соответствующих запросу,

бот последовательно отправляет пользователю информацию о них. К каждому объявлению бот предоставляет две кнопки: "отправить сообщение" и "следующий". При нажатии на кнопку "отправить сообщение" бот предлагает пользователю возможность отправить сообщение владельцу объявления. При выборе кнопки "следующий" бот предлагает пользователю ознакомиться с другим объявлением, соответствующим заданным критериям.

При выборе пункта "Моё объявление" бот инициирует SQL-запрос к базе данных с целью проверки наличия объявления, связанного с идентификатором пользователя. В случае, если в базе данных не обнаруживается объявления с указанным идентификатором, бот предлагает пользователю создать новое объявление. Если же в базе данных имеется соответствующее объявление, бот отправляет его пользователю.

2.3 Разработка телеграмм бота

Прежде чем приступить к написанию кода, рекомендуется заранее создать базу данных для обеспечения своевременного взаимодействия с ней. Поскольку была выбрана система управления базами данных MySQL Workbench, работа будет осуществляться именно с ней. Для создания базы данных в программе необходимо выбрать на верхней панели опцию "Create a new schema in the connected server", а затем ввести название базы данных в соответствующее поле ввода, выполняемые действия представлены на рис. 2.2.

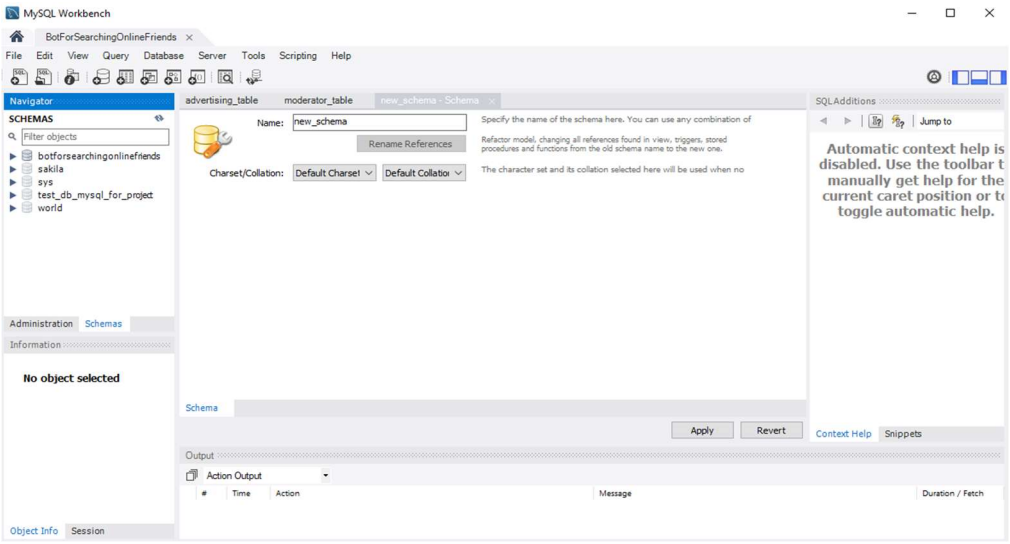

Рис. 2.2 Создание базы данных

Перед началом написания кода необходимо получить токен для будущего телеграмм бота. Для этого следует найти в мессенджере телеграмм бота под именем "BotFather" и отправить ему команду /newbot, после чего ввести имя будущего бота и указать никнейм, по которому можно будет его найти. В ответ на эти действия бот отправит сообщение, содержащее необходимый токен.

Теперь необходимо запустить интегрированную среду разработки IDE PyCharm, выбрать директорию, в которой будет размещён проект, и создать новый проект. В созданной директории следует создать файл с именем main а также файл с именем file_bot, выполняемые действия представлены на рис. 2.3.

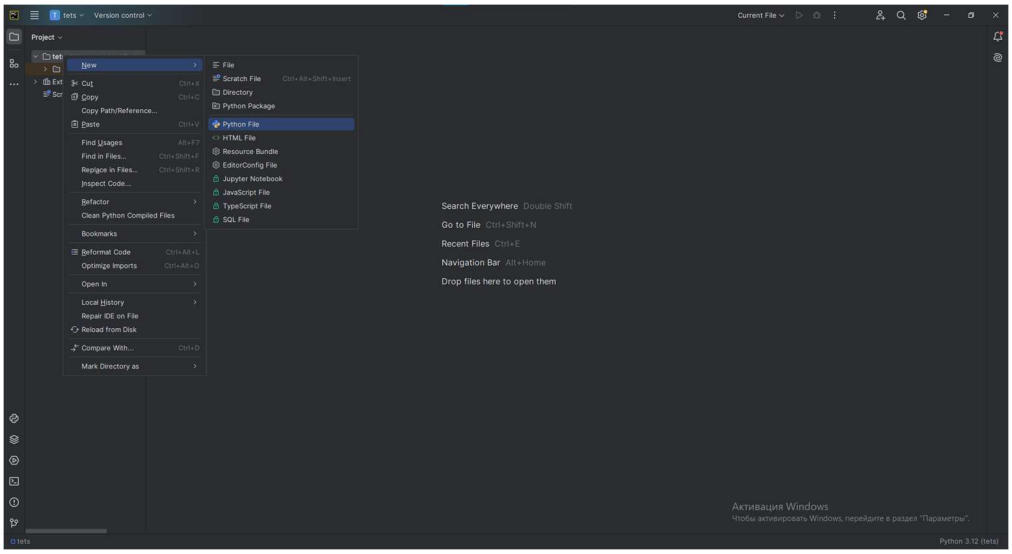

Рис. 2.3 Создание файла main

Далее нужно открыть встроенный терминал и ввести команду "pip3 install telebot". Данная команда установит библиотеку telebot, которая будет использоваться для разработки телеграмм бота, выполняемые действия представлены на рис. 2.4.

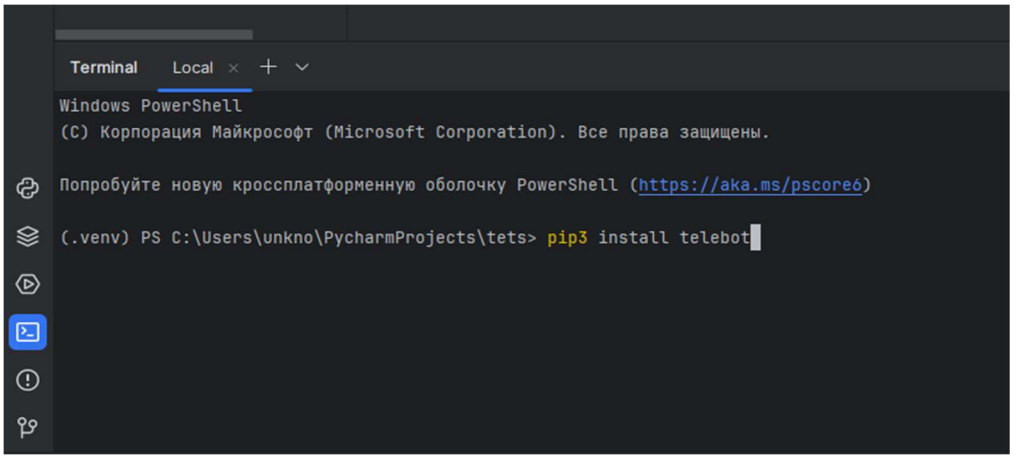

Рис. 2.4 Установка библиотеки telebot.

Теперь, после создания файла и установки библиотеки telebot, можно перейти к написанию кода. В первую очередь необходимо открыть созданный файл file_bot, импортировать в него установленную библиотеку telebot, а затем создать переменную и записать в неё токен, полученный через телеграмм бота “BotFather”.

```
import mysql.connector
import telebot
TOKEN = "7548926182:AAHujrriBEI2iymJAsV3ZkCMNx06xJKuHLU"
bot = telebot.TeleBot(TOKEN)
```

Для обеспечения возможности пользователю создавать собственные объявления и просматривать объявления других, необходимо создать таблицу, в которой будут храниться такие данные, как идентификатор пользователя, имя пользователя, текст объявления, путь к фотографии, статус объявления, причина недопуска, возраст, страна и название видеоигры. Чтобы создать таблицу, нужно отправить в базу данных SQL запрос. Так же стоит создать таблицу для зарегистрированных пользователей и таблицу для заблокированных пользователей.

```
cursor.execute("CREATE TABLE IF NOT EXISTS ADS_TABLE(user_id BIGINT PRIMARY KEY NOT NULL, name VARCHAR(255), text VARCHAR(255), photo_way VARCHAR(255), moderation VARCHAR(255), not_pass_ad VARCHAR(255), age INTEGER, country VARCHAR(255), game VARCHAR(255))")
cursor.execute("CREATE TABLE IF NOT EXISTS BANNED_USERS(user_id BIGINT PRIMARY KEY NOT NULL)")
cursor.execute("CREATE TABLE IF NOT EXISTS ID_USER_TABLE(user_id BIGINT PRIMARY KEY NOT NULL)")
```

После того как написали SQL запросы, нужно скомпилировать код чтобы запросы отправились в базу данных, после чего можно завершить выполнение программы, а SQL запросы можно стереть.

Для начала необходимо открыть файл, названный main.py, в котором будет осуществляться обработка сообщений пользователей. В первую очередь следует настроить бота на начало общения по команде "/start". После получения данной команды бот выполнит SQL запрос к базе данных для проверки наличия

идентификатора пользователя в таблице "BANNED_USERS". В случае обнаружения идентификатора бот уведомит пользователя о его блокировке. Если же идентификатор не будет найден, бот выполнит SQL запрос к таблице "ID_USER_TABLE" для проверки регистрации пользователя. В случае отсутствия идентификатора в данной таблице бот отправит пользователю сообщение об отказе от ответственности со стороны разработчика. Если же идентификатор будет найден, бот поприветствует пользователя.

```
@bot.message_handler(commands=['start'])
def start(message):
    print(message.from_user.id)
    cursor.execute(f"SELECT COUNT(1) FROM BANNED_USERS WHERE
user_id=%s", (message.from_user.id,))
    check_ban = cursor.fetchall()
    for i in check_ban:
        if str(i) == "(0,)":
            cursor.execute(f"SELECT COUNT(1) FROM ID_USER_TABLE WHERE
user_id=%s", (message.from_user.id,))
            check_id = cursor.fetchall()
            for i in check_id:
                if str(i) == "(0,)":
                    bot.send_message(message.chat.id, '   Отказ от ответственности..',
reply_markup=file_Markups.markup_no_responsibility)
                else:
                    bot.send_message(message.chat.id, f"Добро пожаловать,
{message.from_user.first_name}!", reply_markup=file_Markups.markup_menu)
        else:
            bot.send_message(message.chat.id, "Вы забанены!")
```

Для того чтобы бот мог реагировать не только на команды, но и на обычные текстовые сообщения, необходимо разработать функцию, которая будет принимать входящие сообщения и обрабатывать их. Создадим функцию, названную text, которая

будет принимать аргумент message. Когда бот получит сообщение от пользователя, он выполнит SQL запрос к базе данных, в таблицу BANNED_USERS, чтобы определить, заблокирован ли данный пользователь. Если идентификатор будет найден, бот уведомит пользователя о том, что он был заблокирован. В случае, если идентификатор не будет обнаружен, бот продолжит обработку входящего сообщения.

```
@bot.message_handler(content_types=['text'])
def text(message):
    cursor.execute(f"SELECT COUNT(1) FROM BANNED_USERS WHERE user_id=%s",(message.from_user.id,))
    check_ban = cursor.fetchall()
    for i in check_ban:
        if str(i) == "(0,)":
        # далее условия
        else:
            bot.send_message(message.chat.id, "Вы забанены!")
```

В функции по обработке сообщений будет несколько условий, которые будут обрабатывать входящие сообщения от пользователя, если сообщение соответствует одному из заданных условий, бот выполнит соответствующее действие в зависимости от полученного текста. Если же полученное сообщение не будет соответствовать ни одному из условий, бот отправит пользователю уведомление о том, что введённое сообщение не может быть обработано.

Первое условие будет обрабатывать сообщение пользователя, в котором он выражает согласие с отказом от ответственности со стороны разработчика. Ранее было отмечено, что при первом взаимодействии пользователя с ботом, бот отправляет уведомление о том, что разработчик не несёт ответственности за информацию, которую пользователь вводит в процессе использования бота. Таким образом, когда бот отправляет сообщение о отказе от ответственности, пользователю будет предложена кнопка с текстом "Я согласен". При нажатии на эту кнопку пользователь отправляет боту сообщение "Я согласен".

Когда бот получает сообщение "Я согласен" от пользователя, срабатывает условие, связанное с этим сообщением. Далее бот выполняет SQL-запрос к базе данных для добавления идентификатора пользователя в таблицу ID_USER_TABLE, после отправит пользователю приветственное сообщение и предоставит пользователю меню состоящее из кнопок.

```
if message.text == "Я согласен":
    bot.send_sticker(message.chat.id,
"CAACAgIAAxkBAAEMTkFnpRwt_4UIYq_g3nh4zRwAC ")
    bot.send_message(message.chat.id, f"Добро пожаловать,
{message.from_user.first_name}!",    reply_markup=file_Markups.markup_menu)
                cursor.execute(f"INSERT INTO ID_USER_TABLE(user_id)
VALUES (%s)", (message.from_user.id,))
```

Второе условие будет обрабатывать сообщение пользователя "Найти объявления", которое будет отправлено с использованием кнопки из предоставленного меню. При получении данного сообщения от пользователя сработает соответствующее условие, и бот запросит у пользователя ввести название игры, по которой будет осуществляться поиск. После этого произойдёт вызов функции, отвечающей за дальнейшее получение критериев для отбора объявлений.

```
elif message.text == "Найти объявления":
    msg = bot.send_message(message.chat.id, "Введите название игры",
reply_markup=file_Markups.delete_markup)
    bot.register_next_step_handler(msg, file_find_ads.func_game)
```

Третье условие будет активироваться при получении от пользователя сообщения "Моё объявление". В момент срабатывания данного условия будет инициирован вызов функции "my_ads" из предстоящего файла с наименованием "file_my_ads". Эта функция будет отвечать за отображение существующего объявления пользов ателя.

```
elif message.text == "Моё объявление":
    file_my_ads.my_ads(message)
```

Четвёртое условие будет активироваться при получении сообщения "Меню". В момент срабатывания данного условия будет отправлен SQL-запрос в базу данных в таблицу MODERATOR_TABLE для проверки, кто инициировал вызов меню. Если идентификатор пользователя присутствует в данной таблице, то для него будет предоставлено расширенное меню, в противном случае будет отображено стандартное меню.

```
elif message.text == "Меню":
    try:
        cursor.execute(f"SELECT COUNT(1) FROM MODERATOR_TABLE
WHERE user_id={message.from_user.id}")
    check_moder = cursor.fetchall()
        for i in check_moder:
            if str(i) == "(0,)":
                bot.send_message(message.chat.id,"Меню",
                    reply_markup=file_Markups.markup_menu)
            else:
                bot.send_message(message.chat.id, f"Админ - Меню",
reply_markup=file_Markups.markup_admin_panel )
    except Exception as err:
        print(err)
```

Пятое условие активируется при получении сообщения "Создать объявление". В момент срабатывания данного условия будет отправлен SQL-запрос в базу данных, обращающийся к таблице ADS_TABLE с целью проверки наличия объявления, связанного с идентификатором пользователя. Если такое объявление будет обнаружено, бот уведомит пользователя о том, что данное объявление уже существует. В ином случае будет выполнен SQL-запрос на добавление новой записи в таблицу ADS_TABLE с идентификатором пользователя.

```
elif message.text == "Создать объявление":
    cursor.execute(f"SELECT COUNT(1) from ADS_TABLE WHERE user_id=%s",
    (message.from_user.id,))
```

```
t1 = cursor.fetchall()
        for i in t1:
            if str(i) == "(0,)":
                cursor.execute(f"INSERT INTO ADS_TABLE (user_id,
moderation) VALUES (%s, %s)",(message.from_user.id, 'На рассмотрении'))
                msg = bot.send_message(message.chat.id, "Хотите ли добавить
фото к объявлению?", reply_markup=file_Markups.markup_photo)
                bot.register_next_step_handler(msg,
file_Create_ad.start_create_ad)
            else:
                bot.send_message(message.chat.id, "Вы уже создали
объявление!", reply_markup=file_Markups.markup_go_menu)
                print(message.from_user.id)
```

Кроме того, будут разработаны специальные условия для модераторов. Одним из таких условий станет условие, активирующееся при получении сообщения "Запросы ads". В рамках данного условия будет вызываться функция Moderation из файла file_moderation. При вызове данной функции поочередно будут показаны объявления, подлежащие проверке модератором. В случае, если объявление не пройдет проверку, модератор направит сообщение с причиной отказа в публикации, которое бот переотправит пользователю. Если же объявление будет одобрено модератором, оно получит статус "Активно".

```
elif 'Запросы ads' in message.text:
    file_Moderation.Moderation(message)
```

Другим специальным условием для модераторов, которое будет активироваться, станет условие, срабатывающее при получении сообщения “!Все!объявления”. В рамках данного условия будет отправляться SQL-запрос в таблицу ADS_TABLE для извлечения всех существующих объявлений.

```
elif message.text == "!Все!объявления!":
    try:
        cursor.execute(f"SELECT COUNT(1) from ADS_TABLE")
```

```
a123 = cursor.fetchall()
for i in a123:
    if str(i) == "(0,)":
        bot.send_message(message.chat.id, "Объявлений нет")
    else:
        cursor.execute(f"SELECT * FROM ADS_TABLE")
        a = cursor.fetchall()
        for i in a:
            photo = open(f"{i[3]}", "rb")
            bot.send_photo(message.chat.id, photo=photo, caption=
                f"- Имя: {i[1]}\n"
                f"- Возраст: {i[6]}\n"
                f"- Страна: {i[7]}\n"
                f"- Описание: {i[2]}\n"
                f"- Назвние игры: {i[8]}\n"
                f"- Статус: {i[4]}")
```

Следующее условие будет активироваться при получении сообщения "Удалить". В рамках данного условия будет выполнен SQL запрос в базу данных, обращающийся к таблице ADS_TABLE с целью удаления данных, связанных с идентификатором пользователя. Кроме того, будет осуществлено удаление изображения на устройстве, если в таблице имеется путь к данному изображению.

```
elif message.text == "Удалить":
    try:
        cursor.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s",
        (message.from_user.id,))
        a = cursor.fetchall()
        for i in a:
            try:
                os.remove(i[7])
            except:
```

```
                pass
            cursor.execute(f'DELETE FROM ADS_TABLE WHERE
user_id=%s", (message.from_user.id,))
            bot.send_message(message.chat.id, "Ваше объявление удалено!",
reply_markup=file_Markups.markup_go_menu)
    except Exception as err:
        print(err)
```

Теперь необходимо создать файл file_my_ads в той же директории проекта, где располагаются такие файлы, как main.py. Данный файл будет отвечать за отображение объявления, созданного пользователем. Сначала необходимо импортировать файлы file_bot и file_Markups, а также библиотеки mysql.connector и telebot. Затем следует установить соединение с базой данных, создать указатель для работы с таблицами и настроить автоматическое сохранение изменений в таблице.

```
import telebot
from telebot import types
import file_Bot
import mysql.connector
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
```

В этом файле следует реализовать функцию, которая будет вызываться при срабатывании условия из файла main.py на сообщение “Моё объявление”. В рамках данной функции будет отправляться SQL-запрос в таблицу ADS_TABLE для проверки наличия объявлений, созданных пользователем. В случае, если объявление не будет найдено, бот уведомит пользователя об этом и предложит создать новое

объявление. Если же объявление будет найдено, пользователю будет направлено сообщение с данными его объявления.

```
def my_ads(message):
    try:
        cursor.execute(f"SELECT COUNT(1) from ADS_TABLE WHERE user_id=%s", (message.from_user.id,))
        b = cursor.fetchall()
            for i in b:
                if str(i) == "(0,)":
                    bot.send_message(message.chat.id, "У вас нет активных объявлений, вы можете создать!", reply_markup=file_Markups.markup_menu)
                else:
                    cursor.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s", (message.from_user.id,))
                    a = cursor.fetchall()
                        for i in a:
                            photo = open(f"{i[3]}", "rb")
                            bot.send_photo(message.chat.id, photo=photo,
                            f"- Имя: {i[1]}\n"
                            f"- Возраст: {i[6]}\n"
                            f"- Страна: {i[7]}\n"
                            f"- Описание: {i[2]}\n
                            f"- Статус: {i[4]}",
reply_markup=file_Markups.markup_delete_ad)
    except Exception as err:
        print(err)
```

Далее необходимо создать файл с именем file_create_ad в той же директории, данный файл будет отвечать за создание объявления посредством опроса пользователя. Сначала следует импортировать в данный файл библиотеку mysql.connector, а также файлы file_bot и file_Markups. После выполнения импорта

библиотеки и указанных файлов нужно установить соединение с базой данных, создать указатель для работы с таблицами и задать настройки для автоматического сохранения внесенных изменений в таблицы.

```
import mysql.connector
import file_Bot
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
```

Необходимо создать функцию start_create_ad, которая будет служить началом процесса создания объявления и принимать сообщения от пользователя в качестве параметров. В рамках данной функции предусмотрено два условия. Первое условие выполняется при получении от пользователя сообщения "Да", в этом случае пользователю будет направлено сообщение с просьбой предоставить фотографию для включения в предстоящее объявление, после чего будет вызвана функция func_photo. Если первое условие не активируется, сработает второе условие, которое выполняется при получении сообщения "Нет", в этом случае пользователю будет отправлено сообщение с просьбой указать имя пользователя, после чего будет вызвана функция func_name.

```
def start_create_ad(message):
if message.text == "Да":
        msg = bot.send_message(message.chat.id, "Отправьте фото...",
        reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, func_photo)
elif message.text == "Нет":
        msg = bot.send_message(message.chat.id, "Как вас зовут?",
```

```
reply_markup=file_Markups.delete_markup)
bot.register_next_step_handler(msg, func_name)
```

Следующая функция будет называться func_photo. Данная функция будет принимать изображение от пользователя и сохранять его в отдельную директорию, а также отправлять SQL-запрос в базу данных для сохранения пути к изображению в таблице ADS_TABLE. В случае, если в предыдущей функции start_create_ad сработало первое условие, которое активирует функцию func_photo, то эта функция направит пользователю запрос с просьбой указать своё имя, после чего будет вызвана функция func_name.

```
def func_photo(message):
    try:
        photo_id = message.photo[-1].file_id
        photo_file = bot.get_file(photo_id)
        download_file = bot.download_file(photo_file.file_path)
        src = "photos/" + photo_file.file_path.replace('photos/', '')
        with open(src, 'wb') as new_file:
            new_file.write(download_file)
        cursor.execute(f"UPDATE ADS_TABLE SET photo_way=%s WHERE
user_id=%s", (f"{str(src)}", message.from_user.id,))
            msg = bot.send_message(message.chat.id, f"Как вас зовут?",
reply_markup=file_Markups.delete_markup)
            bot.register_next_step_handler(msg, func_name)
            except:
                msg = bot.send_message(message.chat.id, "Пожалуйста,
отправьте фотографию!")
bot.register_next_step_handler(msg, func_photo)
```

Следующая функция, названная func_name, как и предыдущие, будет принимать сообщение от пользователя в качестве параметра. Эта функция отправляет SQL-запрос в таблицу ADS_TABLE для добавления имени пользователя.

После этого пользователю будет направлено сообщение с просьбой указать свой возраст, после чего будет вызвана функция func_age.

```
def func_name(message):
    cursor.execute(f"UPDATE ADS_TABLE SET name=%s WHERE user_id=%s",
(f"{message.text}", message.from_user.id,))
    db.commit()
    msg = bot.send_message(message.chat.id, "Укажите свой возраст")
    bot.register_next_step_handler(msg, func_age)
```

Функция func_age будет осуществлять проверку на то, является ли полученный аргумент числом. В случае, если аргумент не представляет собой число, пользователю будет отправлено сообщение с просьбой ввести числовое значение. Если же аргумент окажется числом, он будет подвергнут проверке на соответствие двум условиям: значение не должно превышать шестьдесят и не должно быть меньше пяти. Если одно из этих условий будет выполнено, пользователю будет направлено сообщение с просьбой указать настоящий возраст. При успешном прохождении аргументом проверки на максимальное и минимальное значение, будет отправлен SQL-запрос для добавления аргумента в таблицу ADS_TABLE в качестве возраста пользователя, после чего будет направлено сообщение с просьбой ввести текст для объявления, далее будет вызвана функция func_game.

```
def func_age(message):
    age = message.text
    if not age.isdigit():
        msg = bot.send_message(message.chat.id, "Пожалуйста, введите число!")
        bot.register_next_step_handler(msg, func_age)
    elif int(age) > 60:
        msg = bot.send_message(message.chat.id, "Введите настоящий возраст!")
        bot.register_next_step_handler(msg, func_age)
    elif int(age) < 10:
        msg = bot.send_message(message.chat.id, "Введите настоящий возраст!")
        bot.register_next_step_handler(msg, func_age)
```

```
        else:
            cursor.execute(f"UPDATE ADS_TABLE SET age=%s WHERE
user_id=%s", (int(message.text), message.from_user.id,))
            msg = bot.send_message(message.chat.id, "Введите текст")
            bot.register_next_step_handler(msg, func_game)
```

Следующая функция, именуемая func_game, выполняет отправку SQL-запроса в таблицу ADS_TABLE с целью добавления ранее введённого пользователем сообщения, содержащего текст к объявлению. После этого пользователю направляется сообщение с просьбой указать название видеоигры, после чего вызывается функция func_text.

```
def func_game(message):
    cursor.execute(f"UPDATE ADS_TABLE SET text=%s WHERE user_id=%s",
    f"{message.text}", message.from_user.id,))
    msg = bot.send_message(message.chat.id, "Введите название игры")
    bot.register_next_step_handler(msg, func_text)
```

Далее будет представлена функция func_text, которая выполняет SQL-запрос к таблице ADS_TABLE с целью добавления в колонку game названия игры, указанного пользователем. После этого отправляется сообщение с просьбой выбрать страну проживания пользователя из предложенных двух стран: Россия и Беларусь, после чего вызывается функция func_final.

```
def func_text(message):
    cursor.execute(f"UPDATE ADS_TABLE SET game=%s WHERE user_id=%s",
    (f"{message.text.lower()}", message.from_user.id,))
    msg = bot.send_message(message.chat.id, "Из какой вы страны?",
    reply_markup=file_Markups.markup_country)
    bot.register_next_step_handler(msg, func_final)
```

Предпоследней функцией является func_final, которая осуществляет отправку SQL-запроса в таблицу ADS_TABLE с целью добавления в колонку country страны, указанной пользователем. После этого выполняется SQL-запрос для получения объявлений, связанных с идентификатором пользователя. Все полученные данные

направляются пользователю для проверки внесённой им информации об объявлении. Пользователю будет предложен выбор из двух кнопок: "верно" и "исправить", далее вызывается функция func_correct_wrong.

```
def func_final(message):
    cursor.execute(f"UPDATE ADS_TABLE SET country=%s WHERE user_id=%s",
    (f"{message.text.lower()}", message.from_user.id,))
    db.commit()
    cursor.execute(f"SELECT * FROM ADS_TABLE WHERE
user_id=%s"(message.from_user.id,))
    a = cursor.fetchall()
    for i in a:
        photo = open(f"{i[3]}", "rb")
        msg2 = bot.send_photo(message.chat.id, photo=photo, caption=
        f"- Имя: {i[1]}\n"
        f"- Возраст: {i[6]}\n"
        f"- Страна: {i[7]}\n"
        f"- Описание: {i[2]}\n"
        f"- Игра: {i[8]}", reply_markup=file_Markups.markup_right_wrong)
        bot.register_next_step_handler(msg2, func_correct_wrong)
```

В последней функции func_correct_wrong будут определены два условия, которые активируются в зависимости от выбора пользователя. Если пользователь нажимает кнопку "верно", срабатывает первое условие, в этом условии пользователю отправляется сообщение "ваше объявление было отправлено на проверку модераторам". В случае, если пользователь решает внести изменения в своё объявление и нажимает кнопку "исправить", активируется второе условие, в котором запрашивается у пользователя ввод его имени. После этого будет вызвана функция func_name, позволяющая пользователю заново ввести все необходимые данные.

```
def func_correct_wrong(message):
    if message.text == "Верно":
        bot.send_message(message.chat.id, "Ваше объявление отправлено на
```

модерацию...",reply_markup=file_Markups.markup_go_menu)

elif message.text == "Исправить":

msg = bot.send_message(message.chat.id, "Начнем с начала...\nКак вас зовут?",reply_markup=file_Markups.delete_markup)

bot.register_next_step_handler(msg, func_name)

Теперь создаём файл под именем file_Moderation в той же директории, где расположены остальные файлы. В созданном файле необходимо импортировать библиотеки mysql.connector и traceback, после чего следует импортировать файлы file_bot и file_Markups. Далее, аналогично предыдущему файлу, требуется установить подключение к базе данных и создать указатель на таблицы.

```
import file_Bot
import mysql.connector
import traceback
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
```

Теперь необходимо разработать функцию, которая будет отправлять SQL-запрос в базу данных в таблицу ADS_TABLE с целью проверки наличия объявлений с статусом "на рассмотрении". В случае, если объявления не будут обнаружены, будет отправлено сообщение "объявлений не найдено". В противном случае будет выполнен SQL-запрос к той же таблице для получения существующих объявлений. Затем создаётся переменная row, в которую добавляется указатель с внутренней функцией, предназначенной для последовательной обработки полученных объявлений. Далее вызывается предстоящая функция search, при этом в качестве параметров передаются переменная row и сообщения, полученные от пользователя.

```
def Moderation(message):
    try:
```

```
            cursor.execute("SET AUTOCOMMIT = 1")
            cursor.execute("SELECT COUNT(1) FROM ADS_TABLE WHERE moderation=%s",('На рассмотрении',))
            find_ads = cursor.fetchall()
            for i in find_ads:
                if str(i) == "(0,)":
                    bot.send_message(message.chat.id, 'Объявлений не найдено')
                else:
                    cursor.execute("SELECT * FROM ADS_TABLE WHERE moderation='На рассмотрении")
                    row = cursor.fetchone()
                    search(message, row)
    except Exception as err:
        print(err)
```

Далее разрабатывается функция search, которая будет отвечать за отображение объявлений в виде сообщений для модераторов. Из полученного аргумента row, содержащего данные объявлений в виде массива, будут извлекаться данные по их индексам для добавления в сообщение, которое будет содержать данные об объявлении, модератору будет представлены три кнопки: “принять”, “отклонить” и “бан”. После этого будет вызвана предстоящая функция solution, в параметры которой передаются указатель и первый объект из аргумента row, представляющий собой идентификатор пользователя, к которому привязано данное объявление.

```
def search(message, row):
    try:
        photo = open(f"{row[3]}", "rb")
            msg = bot.send_photo(message.chat.id, photo=photo, caption=
                    f"- Имя: {row[1]}\n"
                    f"- Возраст: {row[6]}\n"
                    f"- Страна: {row[7]}\n"
                    f"- Описание: {row[2]}\n"
```

```
                f"- Игра: {row[8]}",
                reply_markup=file_Markups.markup_moderation)
            bot.register_next_step_handler(msg, solution, row[0], cursor)
        except Exception as e:
            bot.send_message(message.chat.id,"Ошибка",reply_markup=file_Mar
            kups.markup_menu)
            print('Ошибка:\n', traceback.format_exc())
```

Следующей функцией будет solution, которая принимает три параметра: сообщение от пользователя, идентификатор и указатель. Эта функция отвечает за дальнейшую судьбу объявления в соответствии с решением модератора. В функции предусмотрено три условия, каждое из которых выполняет определённое действие с объявлением и пользователем. В случае, если модератор нажимает кнопку "принять", будет отправлен SQL-запрос в таблицу ADS_TABLE с целью изменения статуса объявления на "Активно". Основателю данного объявления будет направлено сообщение о том, что его объявление прошло проверку. После этого вызывается функция ads, в параметры которой передаётся указатель. Если модератор выберет кнопку "отклонить", будет отправлено сообщение модератору с просьбой указать причину отказа в публикации объявления. После чего будет вызвана функция non_admission в параметры которой передаётся идентификатор и указатель. Если же модератор нажимает кнопку "бан", основателю объявления будет отправлено сообщение о том, что он был заблокирован. Далее выполняются SQL-запросы: первый запрос направляется в таблицу BANNED_USERS для внесения идентификатора заблокированного пользователя, второй запрос отправляется в таблицу ADS_TABLE для удаления объявления заблокированного пользователя, а третий запрос отправляется в таблицу ID_USER_TABLE для удаления идентификатора заблокированного пользователя. После этого вызывается функция ads, в параметры которой передаётся указатель.

```
def solution(message, id, cursor):
    if message.text == "Принять":
        cursor2.execute(f"UPDATE ADS_TABLE SET moderation=%s WHERE
```

```
user_id=%s", ("Активно", int(id),))
            bot.send_message(message.chat.id, "Объявление принято!")
            bot.send_message(id, "Поздравляем! объявление прошло проверку!")
            ads(message, cursor)
        elif message.text == "Отклонить":
            msg = bot.send_message(message.chat.id, "Причина недопуска
объявления:...", reply_markup=file_Markups.delete_markup)
            bot.register_next_step_handler(msg, non_admission, id, cursor)
        elif message.text == "Бан":
            try:
                bot.send_message(id, "Вы были забанены!")
                bot.send_message(message.chat.id, "Был выдан бан")
                cursor2.execute(f"INSERT INTO BANNED_USERS (user_id)
VALUES (id)")
                cursor2.execute(f"DELETE FROM ADS_TABLE WHERE
            user_id=%s", (id,))
                cursor2.execute(f"DELETE FROM ID_USER_TABLE WHERE
            user_id= id ")
                ads(message, cursor)
            except Exception as err:
                print(err)
```

Функция non_admission разработана для обработки ситуаций, в которых объявление пользователя не прошло модерацию. Она принимает три параметра: message, представляющее собой сообщение, содержащие текст причины отказа; id, обозначающее идентификатор пользователя, чье объявление было отклонено; и cursor, выступающее в роли объекта курсора, используемого для выполнения операций с базой данных. Внутри функции реализован блок обработки исключений, что обеспечивает надежность выполнения кода. На первом этапе осуществляется выполнение запроса к базе данных с применением метода execute объекта cursor2, в результате чего происходит обновление статуса модерации объявления в таблице

ADS_TABLE для конкретного пользователя. Статус выставляется на значение, которое описывает причину отклонения, извлекаемую из текста сообщения.По завершении запросов метод send_message объекта bot отправляет пользователю уведомление о том, что его объявление было отклонено, с указанием конкретной причины, содержащейся в сообщении. Под конец происходит вызов функции ads в параметры которой передаётся курсор.

```
def non_admission(message, id, cursor):
    try:
        cursor2.execute(f"UPDATE ADS_TABLE SET moderation=%s WHERE user_id=%s",(f"Отклонено по причине: {message.text}", int(id),))
        cursor2.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s", (int(id),))
        bot.send_message(id, f"Ваше объявление было недопущено по причине: {message.text}")
        ads(message, cursor)
        except Exception as err:
            print(err)
```

Последняя функция ads осуществляет ряд операций. Она принимает два параметра сообщение, полученное от пользователя и объект курсора для взаимодействия с базой данных. Метод fetchone() курсора позволяет извлечь одну строку данных из текущего результата запроса к базе данных, а полученный результат сохраняется в переменной row. После извлечения строки функция проверяет её существование, то есть наличие данных в переменной row. Если данные имеются, происходит вызов функции search, которая принимает в качестве аргументов сообщение и извлечённую строку данных. Данная функция предназначена для обработки или отображения информации, связанной с найденным объявлением. В случае отсутствия данных в row, функция отправляет пользователю уведомление о том, что больше объявлений нет.

```
def ads(message, cursor):
    row = cursor.fetchone()
```

```
    if row:
        search(message, row)
    else:
        bot.send_message(message.chat.id, 'Больше объявлений нет',
reply_markup=file_Markups.markup_go_menu)
```

Теперь создаётся файл file_find_ads в той же директории, где расположен файл main.py. Данный файл будет отвечать за просмотр объявлений других пользователей, данный файл схож по работе с файлом file_Moderation. Для начала необходимо импортировать библиотеку mysql.connector и traceback, после чего следует импортировать файлы file_bot и file_Markups. После выполнения импорта указанных библиотек и файлов необходимо установить соединение с базой данных, создать указатель для работы с таблицами и задать настройки для автоматического сохранения внесённых изменений в таблицы.import mysql.connector.

```
import file_Bot
import file_Markups
import traceback
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
```

Далее создается функция func_start, данная функция вызывается в файле main.py. Функция func_start очищает данные пользователя в базе данных и отправляет сообщение с просьбой ввести название игры, регистрируя обработчик для следующего шага диалога.

```
def func_start(message):
   cursor.execute(f"UPDATE criteries SET game=Null, country=Null, age=Null WHERE
id={message.from_user.id}")
```

msg = bot.send_message(message.chat.id, "Введите название игры", reply_markup=file_Markups.delete_markup)
bot.register_next_step_handler(msg, func_game)

Теперь создаётся первая функция func_game, в качестве параметра в неё передаётся сообщение пользователя, содержащее название видеоигры, по которой будет осуществляться поиск. В данной функции, при получении сообщения пользователя в качестве аргумента, добавляется в таблицу criteries в поле game, куда вносится этот аргумент. К нему применяется встроенная функция lower, что позволяет преобразовать текст в нижний регистр, таким образом упрощая поиск объявлений с использованием SQL-запросов. После этого пользователю отправляется сообщение с просьбой указать возраст, до которого будет осуществляться поиск; полученное сообщение передаётся в качестве аргумента в вызываемую функцию func_age.

```
def func_game(message):
    cursor.execute(f"UPDATE criteries SET game='{message.text.lower()}'")
    msg = bot.send_message(message.chat.id, "До какого возраста искать друга?")
    bot.register_next_step_handler(msg, func_age)
```

Следующая функция func_age предназначена для обработки возрастной информации, введенной пользователем в контексте взаимодействия с ботом. Принимаемый параметр message содержит текстовое сообщение от пользователя, в котором указывается возраст. Сначала происходит извлечение текста этого сообщения и его сохранение в переменной age которая создается внутри функции. Далее выполняется проверка на то, является ли введенное значение числом, с помощью метода isdigit(). Если введенное значение не проходит проверку, бот отправляет пользователю сообщение с просьбой выбрать число из предложенной клавиатуры или указать возраст самостоятельно. Для этой цели используется метод send_message объекта bot, и в качестве параметра reply_markup передается раскладка клавиатуры file_Markups.markup_ages, которая содержит доступные возрастные варианты. В этом случае также регистрируется следующий шаг обработки, назначая

функцию func_age в качестве обработчика для получения следующего ввода от пользователя. Если же введенное значение является корректным числом, оно преобразуется в целое число и сохраняется в таблице criteries в поле age. Затем пользователю отправляется новое сообщение, в котором содержится запрос о стране, откуда следует искать объявления. Снова используется метод send_message, при этом указываются как содержимое сообщения, так и раскладка клавиатуры file_Markups.markup_country для упрощения выбора. Напоследок происходит регистрация следующего шага обработки с помощью метода register_next_step_handler, где в качестве обработчика назначается функция func_country.

```
def func_age(message):
    age = message.text
    if not age.isdigit():
        msg = bot.send_message(message.chat.id, "Пожалуйста, выберите число из предоставленной клавиатуры или укажите самостоятельно!",
    reply_markup=file_Markups.markup_ages)
    bot.register_next_step_handler(msg, func_age)
    else:
        cursor.execute(f"UPDATE criteries SET age={int(message.text)}")
        msg = bot.send_message(message.chat.id, "Из какой страны искать
              объявления?\n(Пожалуйста, пишите без ошибок!)",
                    reply_markup=file_Markups.markup_country)
        bot.register_next_step_handler(msg, func_country)
```

Функция func_country предназначена для обработки выбора пользователя относительно страны поиска объявлений. Входным параметром является message, содержащий текст, в котором пользователь указывает страну. Сначала осуществляется преобразование текста в нижний регистр, после чего значение сохраняется в таблицу criteries в поле country.

Затем функция выполняет SQL-запрос к базе данных, чтобы подсчитать количество объявлений, соответствующих заданным критериям: стране, возрасту,

статус модерации “активно” и наличию искомой игры в соответствующих полях. Результат запроса сохраняется в переменной a с помощью метода fetchall().В последующем идет цикл по полученным результатам, где проверяется, есть ли объявления, соответствующие критериям. Если ничего не будет найдено, то пользователю отправляется сообщение о том, что объявления не найдены. Если объявления имеются, выполняется повторный SQL-запрос для извлечения всех соответствующих данных из таблицы ADS_TABLE, а затем результаты передаются в функцию search для дальнейшей обработки. Весь процесс включает конструкцию try-except для отлова исключений, что позволяет предотвратить сбой программы в случае возникновения ошибок во время выполнения запросов к базе данных.

```
def func_country(message):
    try:
        cursor.execute(f"UPDATE criteries SET country='{message.text.lower()}'")
        cursor.execute(f"SELECT * FROM criteries WHERE id={message.from_use
    r.id}")
        data = cursor.fetchall()
        cursor.execute(f"SELECT COUNT(1) FROM ADS_TABLE WHERE
        country='{ data[0][2]}' AND age<{ data[0][3] + 1}AND moderation=
'Активно' AND game LIKE '%{ data[0][1]}%' OR text LIKE '%{ data[0][1]}%'"
        a = cursor.fetchall()
        for kol_vo_ads in a:
            if str(kol_vo_ads) == "(0,)":
                bot.send_message(message.chat.id, "К сожалению, ничего не
        найдено по вашим критериям :(",
        reply_markup=file_Markups.markup_menu)
            else:
                cursor.execute(f"SELECT * FROM ADS_TABLE WHERE
                country='{data[0][2}' AND age<{data[0][3] + 1} AND
            moderation='Активно' AND game LIKE '%{ data[0][1}%
```

```
                OR text LIKE '%{ data[0][1}%'")
                row = cursor.fetchone()
                search(message, row)
    except Exception as err: print(err)
```

Следующая функция search предназначена для отображения информации об объявлении пользователю. Она принимает два параметра: message, содержащий данные о сообщении пользователя, и row, представляющий собой строку данных, извлеченных из базы данных. В начале функции происходит попытка открыть изображение, используя путь, указанный в третьем элементе строки row[3], что указывает на путь расположение изображения. Следующий шаг включает отправку фотографии пользователю с помощью метода send_photo, где в качестве параметров передаются идентификатор чата, в котором просматривается объявления, открытое фото и текстовое сообщение, составленное в формате многострочного текста, содержащего такие параметры, как имя, возраст, игра, страна и описание, извлекаемые из массива row. Кроме того, добавляется раскладка клавиатуры file_Markups.markup_choose. После отправки сообщения функция регистрирует следующий шаг обработки с помощью метода register_next_step_handler, который передает управление функции like_or_not_like, передавая в качестве параметров идентификатор объявления row[0] и объект cursor для выполнения последующих действий в базе данных.

```
def search(message, row):
        try:
            photo = open(f"{row[3]}", "rb")
            msg = bot.send_photo(message.chat.id, photo=photo, caption=
            f"- Имя: {row[1]}\n"
            f"- Возраст: {row[6]}\n"
            f"- Игра: {row[8]}\n"
            f"- Страна: {row[7]}\n"
            f"- Описание: {row[2]}\n",
            reply_markup=file_Markups.markup_choose)
```

```
            bot.register_next_step_handler(msg, like_or_not_like, row[0], cursor)
        except Exception as e:
            bot.send_message(message.chat.id, "Что-то пошло не так... :(",
            reply_markup=file_Markups.markup_menu)
            print('Ошибка:\n', traceback.format_exc())
```

Следующая функция like_or_not_like принимает три аргумента: сообщение, поступившее от пользователя, идентификатор и курсор. В рамках этой функции осуществляется анализ сообщения пользователя в ответ на отправленное объявление. Если текст сообщения совпадает с фразой “Отправить сообщение”, пользователю будет отправлен вопрос о том, какое сообщение он намеревается отправить. Затем вызывается функция register_next_step_handler, в качестве параметров которой указывается функция send_message_to_user, а также передаются идентификатор и курсор для регистрации следующего шага обработки. В случае, если пользователь отправляет сообщение с текстом “следующий”, будет выполнен вызов функции ads, в параметры которой также передаются сообщение и курсор. Если же ни одно из условий не выполнится, так как пользователь ввёл неподходящий текст, ему будет отправлено предупреждение о необходимости использования предоставленной клавиатуры.

```
def like_or_not_like(message, id, cursor):
    if message.text == "Отправить сообщение":
        msg = bot.send_message(message.chat.id, f"Какое сообщение отправить?",
        reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, send_message_to_user, id, cursor)
    elif message.text == "Следующий":
        ads(message, cursor)
    else:
        bot.send_message(message.chat.id, "пользуйтесь предоставленной
        клавиатурой!", reply_markup=file_Markups.markup_menu)
```

Функция send_message_to_user создана для отправки сообщения от пользователю к владельцу объявления. Данная функции принимает три аргумента,

message сообщение от пользователя, id идентификатор владельца объявления, и курсор. Отправка сообщения осуществляется через вызов функции send_message в параметры которой указывается полученный аргумент id который несёт в себе идентификатор владельца объявления, и полученное сообщение от пользователя, далее вызывается функция ads в параметры которой передаются сообщение и курсор.

```
def send_message_to_user(message, id, cursor):
    bot.send_message(id, f'вам пришло сообщение от
    @{message.from_user.username}\nСодержание: {message.text}')
    ads(message, cursor)
```

Последняя функция ads осуществляет ряд операций. Она принимает два параметра сообщение, полученное от пользователя и объект курсора для взаимодействия с базой данных. Метод fetchone() курсора позволяет извлечь одну строку данных из текущего результата запроса к базе данных, а полученный результат сохраняется в переменной row. После извлечения строки функция проверяет её существование, то есть наличие данных в переменной row. Если данные имеются, происходит вызов функции search, которая принимает в качестве аргументов сообщение и извлечённую строку данных. Данная функция предназначена для обработки или отображения информации, связанной с найденным объявлением. В случае отсутствия данных в row, функция отправляет пользователю уведомление о том, что больше объявлений нет.

```
def ads(message, cursor):
    row = cursor.fetchone()
    if row:
        search(message, row)
    else:
        bot.send_message(message.chat.id, 'Больше объявлений нет',
reply_markup=file_Markups.markup_go_menu)
```

Далее создается файл show_advertising, данный файл предназначен для управления показом рекламы в рамках бот-приложения, взаимодействующего с пользователями. Основной целью его существования является извлечение и

демонстрация рекламных объявлений из базы данных, а также отправка соответствующих сообщений пользователям в чатах. Файл включает в себя такие ключевые элементы, как подключение к базе данных MySQL, установка курсора для выполнения запросов, а также использование файла file_Bot для взаимодействия с ботом и файла file_Markups для применения клавиатурных разметок.

```
import random
import file_Bot
import mysql.connector
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
```

В функции show_advertising реализовано получение случайного идентификатора для выбора рекламного объявления, а затем выполняется запрос к таблице advertising_table, чтобы извлечь данные рекламы. Если изображение рекламы доступно, оно отправляется пользователю в виде фото с соответствующим описанием, в противном случае пользователю отправляется текстовое сообщение с содержанием объявления. Вся логика работы функции обернута в блок обработки исключений для управления возникшими ошибками, что обеспечивает стабильность работы бота. Таким образом, файл выполняет важную функцию в системе, позволяя эффективно интегрировать рекламные материалы в процессы общения с пользователями.

```
def show_advertising(message):
    try:
        cursor.execute(f"SELECT COUNT(*) FROM advertising_table")
        check_adv = cursor.fetchone()
        for i in check_adv:
```

```
            ran = random.randint(1, int(i))
            cursor.execute(f"SELECT * FROM advertising_table WHERE id={ran}")
            data_advertising = cursor.fetchall()
            for i in data_advertising:
                if i[4] == None:
                    bot.send_message(message.chat.id, f"{i[2]}\n{i[3]}",
                    reply_markup=file_Markups.markup_menu)
                else:
                    photo = open(f"{i[4]}", "rb")
                    bot.send_photo(message.chat.id, photo=photo,
caption=f"{i[2]}\n{i[3]}",
        reply_markup=file_Markups.markup_menu)
        except Exception as err:
            print(err)
```

Последний файл file_Markups.py служит для создания и настройки клавиатурных разметок, Основная цель файла заключается в формировании различных пользовательских интерфейсов, позволяющих пользователю взаимодействовать с ботом через кнопки. Перед началом настройки пользовательской клавиатуры, в данный файл импортируется модуль types из библиотеки telebot.

Пользовательская клавиатура меню.

```
markup_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_create = types.KeyboardButton("Создать объявление")
btn_find = types.KeyboardButton("Найти объявления")
btn_my_ad = types.KeyboardButton("Моё объявление")
markup_menu.add(btn_create, btn_find, btn_my_ad)
markup_go_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_menu = types.KeyboardButton('Меню')
markup_go_menu.add(btn_menu)
```

Пользовательская клавиатура для ответа на запрос добавить изображение к объявлению или нет.

```
markup_photo = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_yes = types.KeyboardButton("Да")
btn_no = types.KeyboardButton("Нет")
markup_photo.add(btn_yes, btn_no)
```

Пользовательская клавиатура для подтверждения правильности объявления или запроса её изменить.

```
markup_right_wrong = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_right = types.KeyboardButton("Верно")
btn_wrong = types.KeyboardButton("Исправить")
markup_right_wrong.add(btn_right, btn_wrong)
```

Пользовательская клавиатура для запроса на удаления объявления.

```
markup_delete_ad = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_delete_ad = types.KeyboardButton("Удалить")
markup_delete_ad.add(btn_delete_ad, btn_menu)
```

Пользовательская клавиатура для выбора возраста владельцев объявлений.

```
markup_ages = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_13 = types.KeyboardButton("13")
btn_16 = types.KeyboardButton("16")
btn_18 = types.KeyboardButton("18")
btn_20 = types.KeyboardButton("20")
btn_25 = types.KeyboardButton("25")
markup_ages.add(btn_13, btn_16, btn_18, btn_20, btn_25)
```

Пользовательская клавиатура созданная для модераторов объявлений, данная клавиатура предназначена для запрос о принятии объявления, отказе его либо же блокировка пользователя.

```
markup_moderation = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_accept = types.KeyboardButton("Принять")
btn_reject = types.KeyboardButton("Отклонить")
```

```
btn_ban = types.KeyboardButton('Бан')
markup_moderation.add(btn_accept, btn_reject, btn_ban)
```

Пользовательская клавиатура созданная для модераторов, данная клавиатура создана для запросов на показ объявлений, дополнена кнопками из обычного меню.

```
markup_admin_panel = types.ReplyKeyboardMarkup(resize_keyboard=True)
text = "Запросы ads"
btn_modering_ads = types.KeyboardButton(text)
btn_admin_show_all_ads = types.KeyboardButton("!Все!объявления!")
markup_admin_panel.add(btn_create, btn_find, btn_my_ad, btn_modering_ads, btn_admin_show_all_ads)
```

Пользовательская клавиатура для отправки сообщения владельцу сообщения, либо же показ следующего объявления.

```
markup_choose = types.ReplyKeyboardMarkup(resize_keyboard=True)
choose1 = types.KeyboardButton("Отправить сообщение")
choose2 = types.KeyboardButton("Следующий")
markup_choose.add(choose1, choose2)
```

Пользовательская клавиатура для соглашения об отказе ответственности со стороны разработчика.

```
markup_no_responsibility = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_no_responsibility = types.KeyboardButton("Я согласен")
markup_no_responsibility.add(btn_no_responsibility)
```

Пользовательская клавиатура для выбора страны проживания.

```
markup_country = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_rus = types.KeyboardButton("Россия")
btn_belrus = types.KeyboardButton("Беларусь")
markup_country.add(btn_belrus, btn_rus)
```

2.4 Размещение телеграмм бота на арендованном сервере

Для развертывания бота для телеграмм лучше всего подходит VDS/VPS. Он не требует наличия домена в обязательном порядке, и к нему можно будет обратиться по выделенному IP-адресу, что удобно при развертывании бота. К тому же такие

серверы лучше поддерживают увеличение масштабов проекта, так что при увеличении потока посетителей не придется переходить на другой сервер, достаточно будет просто сменить тариф или его настройки.

Выбор тарифа обычно обуславливается потоком клиентов, но не сложностью бота. Для развертывания как простенького, так и нейросетевого бота не потребуется много дискового пространства. Нет нужды и в больших вычислительных мощностях, большинству ботов более чем достаточно 512 МБ оперативной памяти. Поэтому можно выбирать самые недорогие тарифы.

Ключевым требованием к хостингу является стабильность подключения, что обуславливает необходимость отказа от shared-хостингов для телеграм-ботов. В данном контексте VPS/VDS предоставляет возможность обеспечения надежного подключения. Рекомендуемой операционной системой для сервера является Ubuntu, поскольку большинство модулей для языков программирования разрабатываются именно для данной платформы. Тарифы на виртуальные серверы данного типа начинаются от 169 рублей в месяц.

Аренда сервера будет осуществляться через компанию Timeweb. После получения VPS необходимо установить соединение с ним по протоколу SSH с использованием специализированного клиента, например, Termius, применяя данные из личного кабинета на хостинге для телеграм-бота. Для этого потребуются следующие параметры: Hostname, Login, Password и Port. Указанные данные вводятся в соответствующие поля нового подключения, выполняемые действия представлены на рис. 2.5, рис. 2.6.

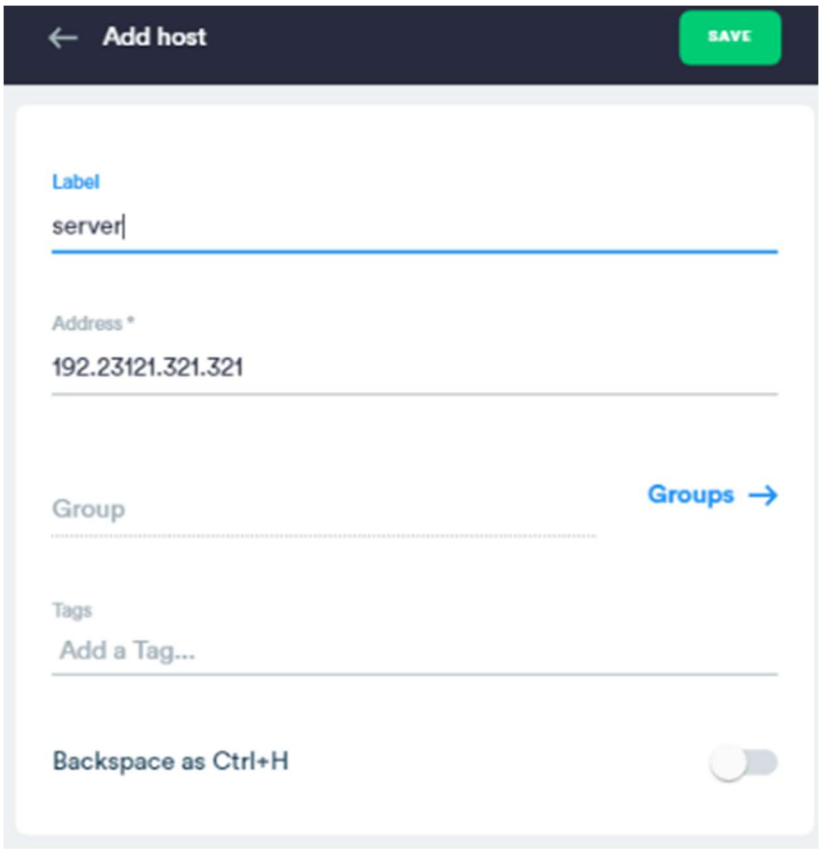

Рис. 2.5 Ввод IP и Hostname

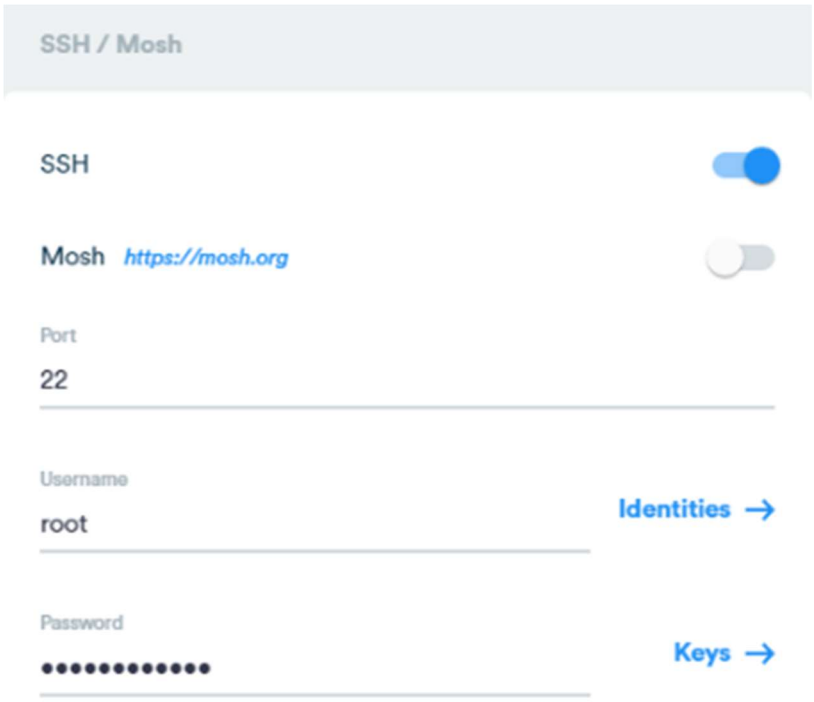

Рис. 2.6 Ввод Port, Login, Password

Сохраняем установленное соединение и возвращаемся на главный экран программы, где отображается наше новое подключение. Затем необходимо нажать на

название сервера, которое было введено ранее, выполняемое действие представлено на рис. 2.7.

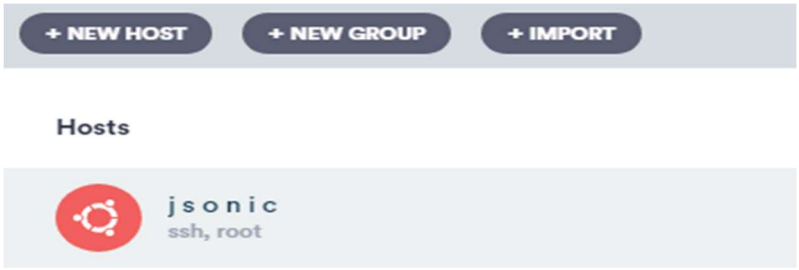

Рис. 2.7 Сохраненное подключение

Процесс завершен, мы получили доступ к консоли сервера, рис. 2.8:

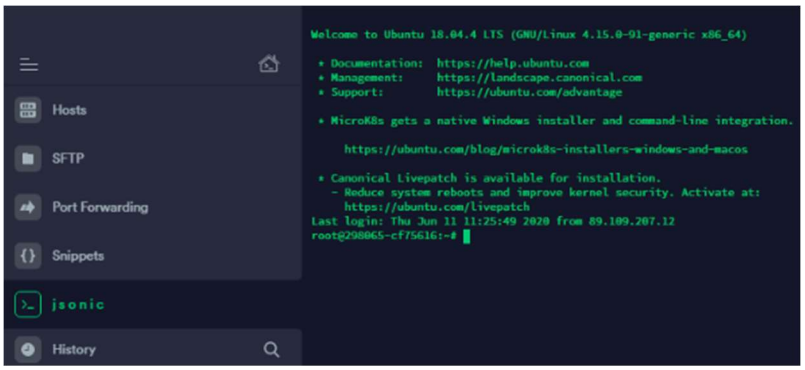

Рис. 2.8 Консоль сервера

Из данной консоли сможем осуществлять управление сервером, загружать необходимые модули и библиотеки. Для обеспечения функционирования бота серверу потребуется модуль языка Python. Поскольку рассматривается бот, разработанный на третьей версии языка, необходимо установить python3. Для этого в консоль сервера следует ввести следующие команды:

sudo apt —reinstall install python3 -y

sudo apt —reinstall install python3-pip –y

Необходимо также учесть библиотеки. Для разработки бота использовалась библиотека telebot. Для её установки следует ввести следующую команду:

pip install telebot

Теперь нужно установить удобный менеджер процессов PM2 и язык программирования NodeJS с менеджером пакетов npm для его работы:

sudo apt install nodejs

sudo apt install npm

npm install pm2 –g

Настройка сервера завершена, и теперь можно отключиться от SSH. Теперь потребуется установить соединение с сервером по протоколу SFTP. Для его настройки необходимо перейти во вкладку SFTP в Termius, нажать на “select host” и выбрать имя сервера. После этого создается папка с произвольным названием на латинице, рис. 2.9.

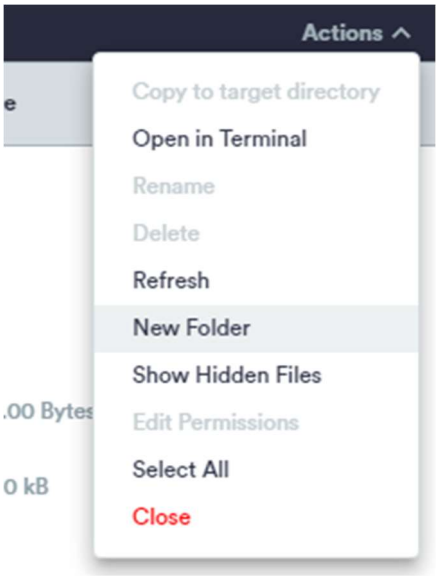

Рис. 2.9 Создание папки в Termius

Затем необходимо скопировать в созданную папку файлы бота. Termius поддерживает функцию перетаскивания drag'n'drop, что позволяет просто переместить содержимое вашей папки с ботом.

Бот готов к запуску. Для этого в консоли сервера последовательно вводим следующие команды:

cd telegram_bot_for_friends

pm2 start main.py --interpreter=python3

Бот успешно запущен и функционирует. Для подтверждения этого факта можно ввести в консоли сервера команду менеджера процессов:

pm2 list

Команда выведет в консоли список процессов, запущенных на сервере. Необходимо найти строку с названием бота. Если в столбце «Status» напротив нее указано «Online», это свидетельствует о том, что бот успешно запущен.

2.5 Проверка работоспособности телеграмм бота

В данной подглаве будет рассмотрена процедура проверки работоспособности телеграмм бота, разработанного в рамках данного проекта. Работоспособность является одним из ключевых аспектов успешного функционирования любого программного продукта, и телеграмм бот не является исключением. Для подтверждения корректности работы всех функций бота будет проведена серия тестов, направленных на обеспечение их соответствия первоначальным требованиям. В качестве метода проверки выбрана визуализация процесса использования бота с помощью скриншотов. Данный подход позволяет наглядно продемонстрировать функционирование бота в реальном времени, а также подтвердить его функциональные возможности.

Начало общения с ботом осуществляется посредством команды /start. Поскольку до этого момента взаимодействие с телеграмм ботом отсутствовало, в ответ на данную команду бот отправит сообщение, информирующее о отказе от ответственности со стороны разработчика “ Создатель бота не несет ответственности за украденные данные пользователя, за распространение информации или любые другие действия, совершенные пользователями с использованием данного бота. Использование бота осуществляется на свой страх и риск. Владелец бота не несет ответственности за какие-либо убытки, ущерб, проблемы или последствия, возникшие в результате использования данного бота или связанные с ним. Пользователь самостоятельно несет ответственность за свои действия.”, а также предоставляет кнопку с текстом “Я согласен”, выполняемое действие представлено на рис. 2.10.

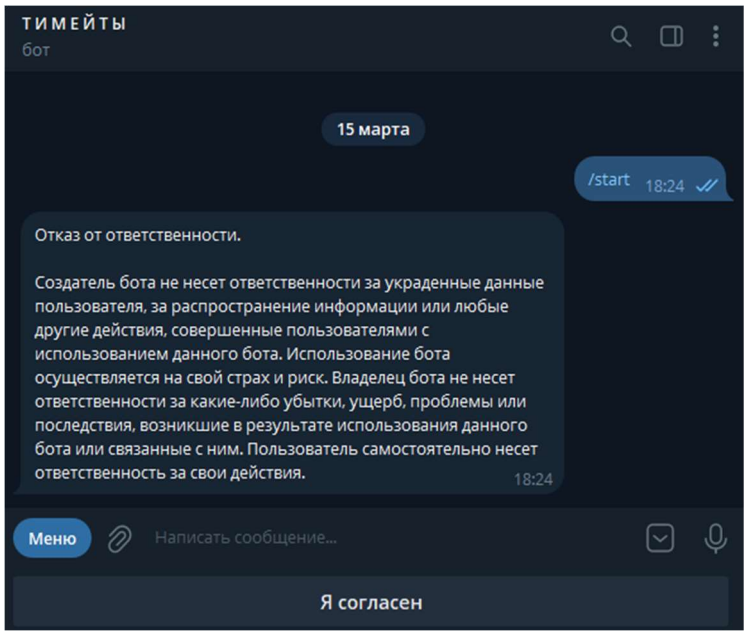

Рис. 2.10 Начало общения с телеграмм ботом

После подтверждения отказа от ответственности со стороны разработчика и активации кнопки, бот отправляет приветственное сообщение, в котором представлено меню в виде клавиатуры. Данное меню состоит из трёх кнопок: "создать объявление", "найти объявление" и "моё объявление", рис. 2.11.

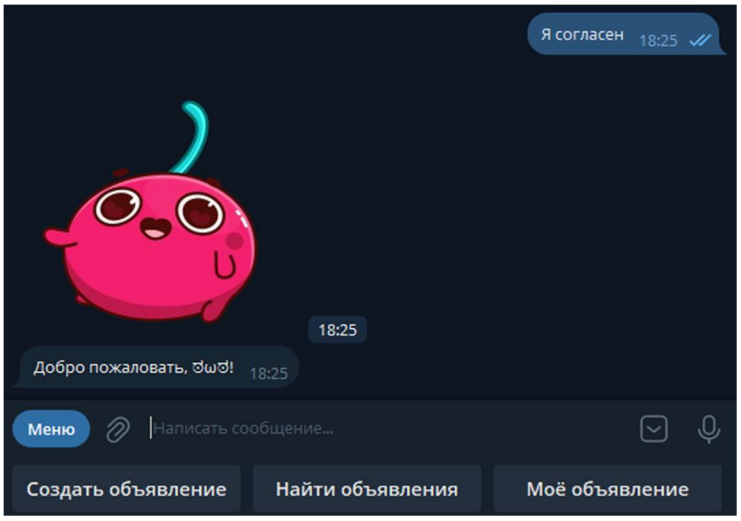

Рис. 2.11 Пользовательское меню

При активации кнопки "моё объявление" бот отправляет сообщение, информирующее о том, что созданного пользователем объявления не существует, и предлагает возможность его создания, рис. 2.12.

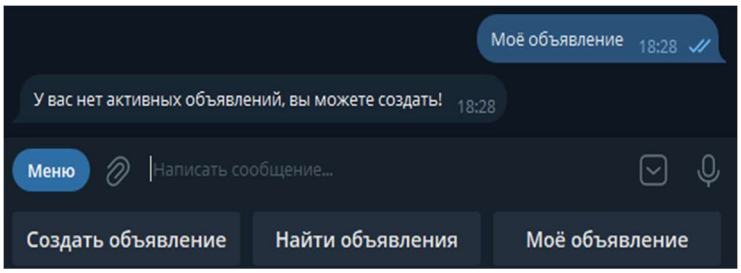

Рис. 2.12 Отсутствие созданного объявления

При активации кнопки "создать объявление" бот инициирует отправку сообщения, в котором задаётся вопрос о необходимости добавления изображения к объявлению. В этом сообщении пользователю представляется клавиатура, содержащая две кнопки: "да" и "нет", рис. 2.13.

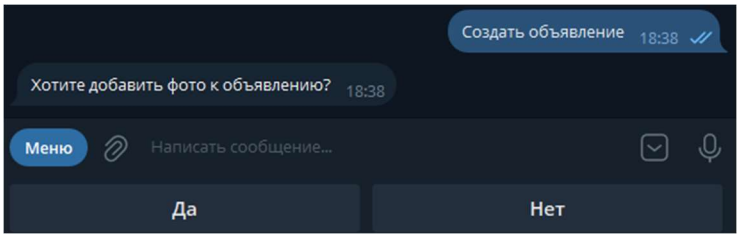

Рис. 2.13 Начало создания объявления

Выбрав кнопку “да” бот запросит изображение, после попросит ввести имя, если выбрать кнопку “нет”, бот попросит ввести имя. Рис. 2.14.

Рис. 2.14 Отправка изображения

После ввода имени, бот просит вести следующие данные: возраст, описание к объявлению, название игры и страну проживания, рис. 2.15.

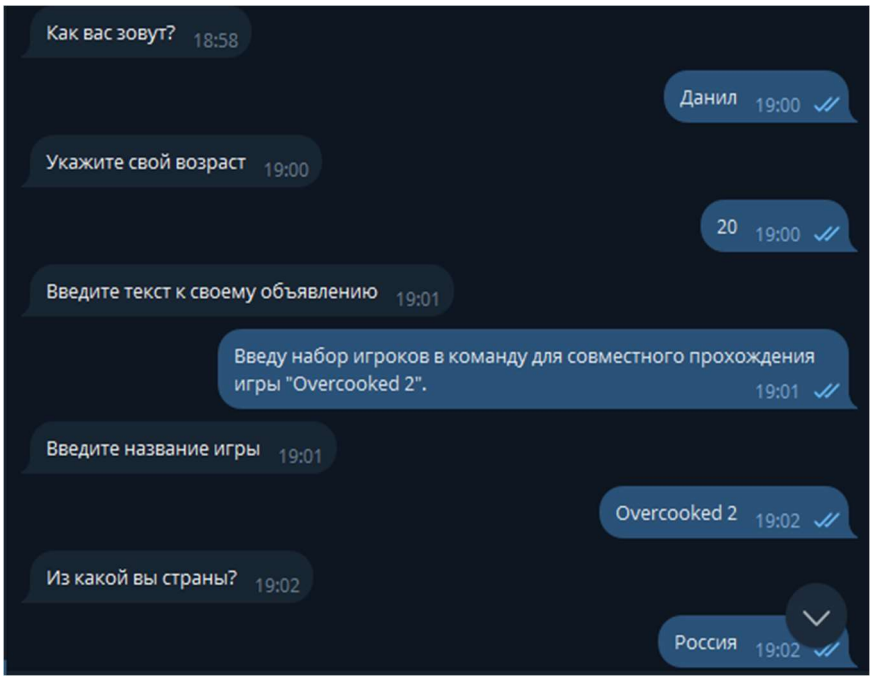

Рис. 2.15 Ввод данных

По завершении ввода запрашиваемых данных бот демонстрирует пользователю, каким образом будет выглядеть итоговое объявление. На данном этапе пользователю предлагается клавиатура с двумя кнопками: "верно" и "исправить".

Эта клавиатура предоставляет возможность внести изменения в случае недовольства, либо согласиться с представленным вариантом и отправить объявление на модерацию, рис 2.16, рис. 2.17.

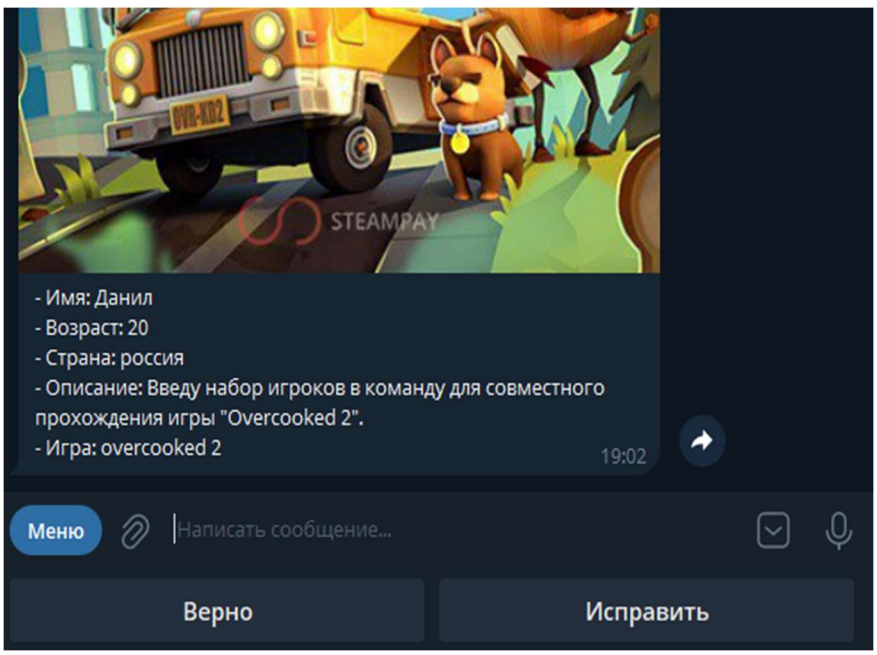

Рис. 2.16 Показ объявления

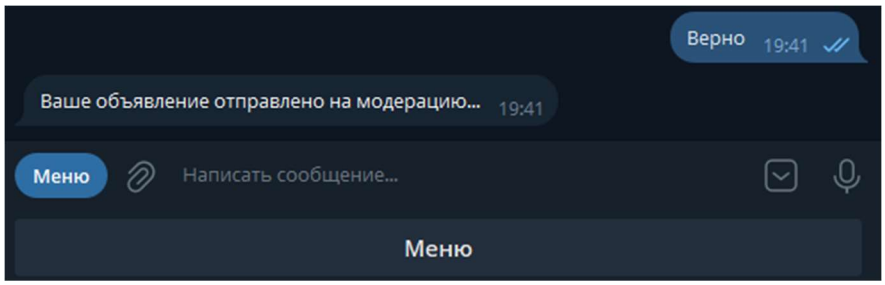

Рис. 2.17 отправка объявления на модерацию

Теперь в случае нажатия на кнопку “моё объявление” в предоставленном меню, бот продемонстрирует созданное объявление со статусом “на рассмотрении”, рис. 2.18. Поскольку объявление не было рассмотрено модератором. После проведения рассмотрения и получения одобрения модератором, объявление приобретет статус “активно”.

Рис. 2.18 Показ созданного объявления

В случае нажатия кнопки “найти объявления”, бот попросит ввести критерии, по которым будет произведен поиск. К числу этих критериев относятся: название игры, минимальный возраст игрока, страна проживания, если же не важно из какой страны искать, то будет представлена кнопка “не имеет значения” выполняемое действие представлено на рис. 2.19, рис. 2.20.

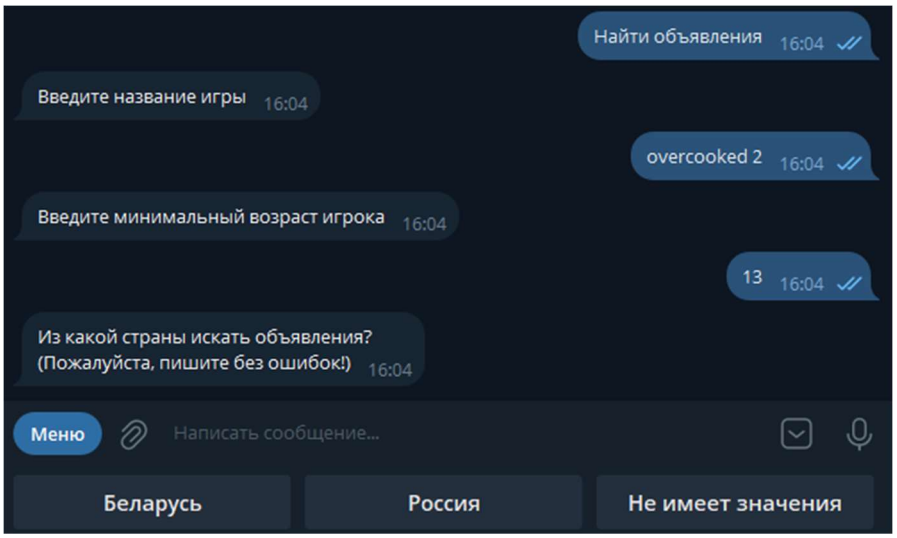

Рис. 2.19 Поиск объявлений

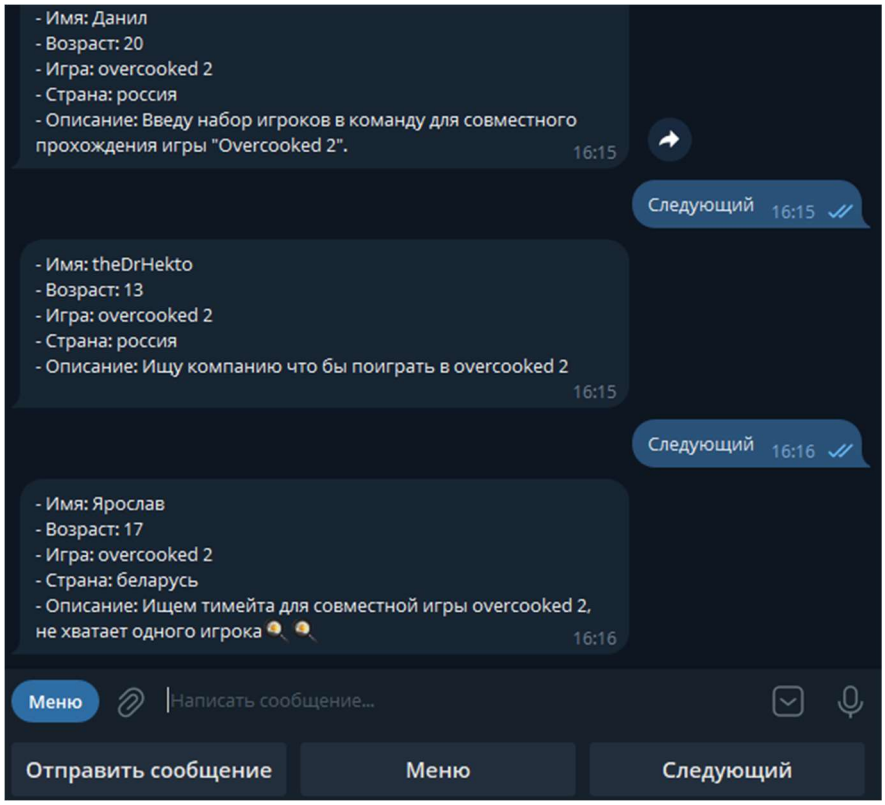

Рис. 2.20 Показ объявлений по полученным критериям

Теперь проведем проверку функциональности телеграмм бота с точки зрения его использования модератором. При начале взаимодействия с ботом в статусе модератора пользователю будет представлено расширенное меню, в которое будут добавлены дополнительные кнопки: "Запросы ads" и "Все объявления", рис. 2.21.

При выборе кнопки "Запросы ads" пользователю будут отображены объявления, требующие модерации. Для модератора будет доступна клавиатура с тремя функциями: "Принять", "Отклонить" и "Бан", рис. 2.22. В случае, если

объявление соответствует установленным требованиям, а именно присутствуют ссылки на сторонние сайты, чаты и каналы, описание написано на русском языке, а также если страна указана как Россия или Беларусь, модератор нажимает кнопку "Принять", что приводит к присвоению объявлению статуса "Активно", рис. 2.23.

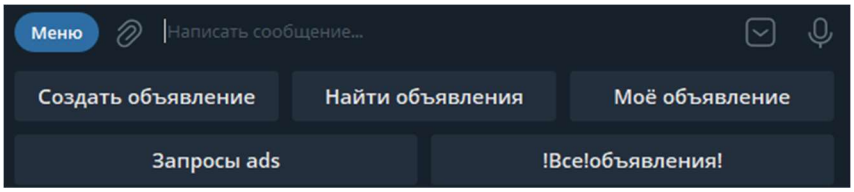

Рис. 2.21 Расширенное меню

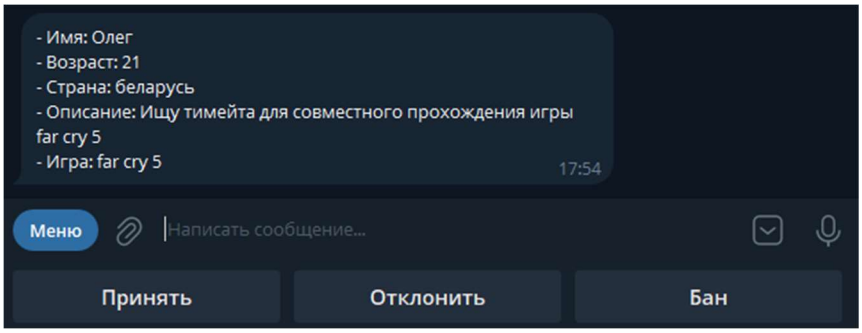

Рис. 2.22 Меню модератора

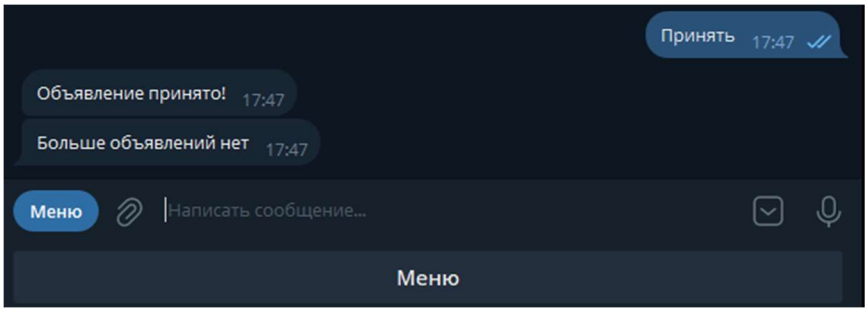

Рис. 2.23 Принятие объявления

В случае, если объявление не соответствует установленным требованиям, модератор имеет возможность нажать кнопку “Отклонить”. После этого бот запрашивает у модератора ввод причины недопуска, рис. 2.24.

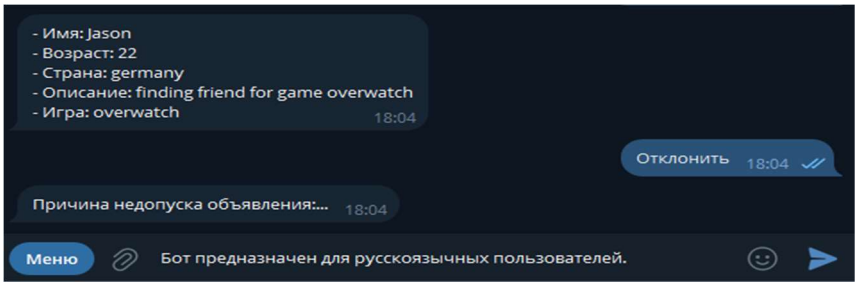

Рис. 2.24 Отклонение объявления

ГЛАВА 3 Экономическая эффективность разработки телеграмм бота

3.1 График выполнения работ

Таблица 1 - Этапы работ.

Этап разработки	Время, часов	Используемое оборудование	Используемое программное обеспечение
1	2	3	4
Анализ требований и проектирование	20	Компьютер	LibreOffice
Написание кода бота (Backend)	110	Компьютер	Python, Телеграмм API, Github,
Разработка базы данных	10	Компьютер	MySQL WorkBrench
Тестирование бота	40	Компьютер	GitHub, Телеграмм
Размещение и настройка на сервере	20	Сервер	Timeweb
Итого	200	---	---

Разработка телеграмм бота заняла 200 часов, что составляет 25 рабочих дней.

Рассчитать стоимость электроэнергии, потребляемой за год можно по формуле:

$$Зэн = Рэвм*Тэвм*Сэл \quad (1)$$

Где

Рэвм - суммарная мощность ЭВМ, кВт;

Тэвм - время эксплуатации компьютера при создании веб-приложения, часов;

Сэл - стоимость одного кВт/ч электроэнергии, руб;

Зэн = 0,3 * 200 * 5,51 = 380 руб.

Итог: Стоимость электроэнергии за время разработки составляет 380 рублей.

3.2 Затраты на разработку

Таблица 2 - Расчёт затрат на оплату труда.

Должность	Часовая тарифная ставка, руб.	Отработано часов	Сумма, руб.
1	2	3	4
Разработчик	250	160	40 000
Тестировщик	250	40	10 000
Итог	---	200	50 000

Пояснения:

- Разработчик выполняет основную часть работы: написание кода и интеграцию компонентов;
- Тестировщик занимается тестированием и устранением ошибок;

Разработчик:

- Количество сотрудников: 1;
- Часовая ставка: 250 руб/час;
- Общее количество часов: 160 час;
- Общая стоимость: 40 000 рублей;
- Расчёт: 1 * (250 * 160) = 40 000 рублей;

Тестировщик:

- Количество сотрудников: 1;
- Часовая ставка: 250 руб/час;
- Общее количество часов: 40 час;
- Общая стоимость: 10 000 рублей;
- Расчёт: 1 * (250 * 40) = 10 000 рублей;

Итоговая стоимость труда: 50 000 рублей.

Таблица 3 - Расчёт стоимости оборудования.

Наименование	Цена за единицу	Стоимость
1	2	3
Компьютер	30 000	30 000
Монитор	4 000	4 000
Клавиатура и мышь	2 500	2 500
Интернет - роутер	3 000	3 000
Итого		39 500

Амортизация за период разработки с учётом срока службы на 3 года составила:

An = (39 500 / 3 / 365) * 25 = 901 рублей.

Расчёт затрат на текущий ремонт техники.

Зтр = (39 500 / 365) * 5% * 25 = 135 рублей.

Таблица 4 - Расчёт затрат на программное обеспечение, используемое в работе.

Наименование	Цена лицензии, подписки	Цена за 1 день использования	Количество дней работы	Стоимость
1	2	3	4	5
Облачный хостинг (Timeweb Cloud)	320	10	365	3 840
Итого	---	---	---	3 840

Таблица 5 - Расчёт затрат на услуги связи и интернета.

Наименование	Тариф, руб/мес.	Кол-во дней использования	Стоимость одного дня, руб.	Стоимость, руб.
1	2	3	4	5
Мобильная связь	1000	365	33	12 045
Интернет	700	365	22	1 400
Итого	---	---	---	13 455

Таблица 6 - Сводная смета затрат на разработку веб-приложения.

Вид затрат	Величина, руб.
1	2
Электроэнергия	380
Затраты на оплату труда	50 000
Амортизация оборудования	901
Затраты на текущий ремонт	153
Прочие затраты	17 285
Итого	68 719

3.3 Расчёт эксплуатационных расходов

Таблица 7 - Эксплуатационные расходы

Вид затрат	Эксплуатационные затраты	
	Состав затрат	Планируемая сумма (срок 1 год)
зарплата управленческого персонала, работающего с использованием программного продукта	Оплата труда за администрирование, контроль за работой бота	360 000
Затраты на программные средства	Оплата хостинга	3 840
Затраты на носители информации	Расходы на физические носители	3 000
Прочие затраты	Непредвиденные расходы, комиссия платёжных систем	25 776
Маркетинг и продвижение	Заказ рекламы	20 000
Итог	---	412 616

Сводная смета + Эксплуатационные расходы = Совокупная стоимость (2)

68 719 + 412 616 = 481 445 руб.

Совокупная стоимость разработки составила: 481 335 руб.

3.4 Расчёт показателей экономической эффективности

При внедрении телеграмм бота, произойдут следующие изменения:

Телеграмм бот полностью автоматизирует процесс подбора друзей на основе заданных пользователем критериев (интересы, геолокация и т. д.). Это позволит сократить время на подбор друзей по индивидуальным критериям.

Удобство использования телеграмм бота в сочетании с его доступностью позволяет расширить базу пользователей. Предположительно, внедрение бота привлечёт 1000 новых пользователей в первый год, что увеличит потенциальный доход проекта через заказ рекламы и платных подписок.

Предположим, что каждый третий пользователь за счёт платной подписки приносит доход 250 руб/мес. Таким образом, количество пользователей с платной подпиской составит: 1000 / 3 = 333 пользователя.

Прибыль от платных подписок: 333 пользователя * 250 руб/мес. = 83 250 руб/мес. А доход от рекламы составит 5000 рублей в месяц.

Ежемесячная прибыль рассчитывается по формуле:

$$ЕЖП = ЕПП + ЕПР \quad (3)$$

Где

ЕП – общая ежемесячная прибыль;

ЕПП - ежемесячная прибыль от подписок;

ЕПР – ежемесячная прибыль от рекламы;

83 250 руб. + 5 000 руб. = 88 250 руб.

Срок окупаемости проекта составит: 481 335 руб. / 88 250 руб. = 5.5 месяцев или примерно 5 месяцев и 2 недели.

Чистая прибыль после окупаемости будет:

Общий доход за 6 месяцев = 88 250 × 6 = 529 500 руб.

Затраты на разработку: 529 500 – 481 335 = 48 165 руб.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной выпускной квалификационной работы была разработана функциональная система — телеграм бот для поиска друзей. Основной целью исследования было создание бота, способного обрабатывать введенные пользователем данные и предлагать актуальные совпадения на основе общих интересов. Для достижения этой цели были поставлены и успешно решены несколько задач, включая анализ проблемы, изучение существующих решений, разработку плана и реализацию самого бота.

В процессе работы использовался язык программирования Python, что позволило эффективно реализовать функционал бота. Также был проведен расчет экономической эффективности проекта, включающий оценку затрат на разработку, энергопотребление, оплату труда, стоимость оборудования и эксплуатационные расходы.

Следует отметить, что в процессе разработки не возникло серьезных проблем, что свидетельствует о высоком уровне подготовки и планирования. Разработанный телеграм бот может быть полезен для определенной аудитории, предоставляя возможность находить друзей на основе общих интересов. В качестве рекомендаций по дальнейшему развитию проекта можно выделить возможность создания более удобного интерфейса для пользователей, а также использование других систем управления базами данных для повышения скорости обработки запросов. Оптимизация кода и изменение дизайна бота также могут значительно улучшить пользовательский опыт.

Таким образом, работа завершена успешной реализацией проекта, который открывает новые перспективы для дальнейших исследований в области разработки интеллектуальных систем для социальных взаимодействий. Полученные результаты могут быть использованы для создания более сложных и функциональных решений в будущем.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Федеральные законы и нормативные документы:

1. – ФЗ РФ «О защите прав потребителей от 07.02.92 № 2300-1 с доп. и измен. от 09.01.17 №2 – ФЗ.»
2. – ФЗ РФ "Об авторском праве и смежных правах" от 09.07.1993 № 5351-1 (последняя редакция)
3. – Приказ МИНФИНА РОССИИ от 17.09.2020 № 204н «Об утверждении Федеральных стандартов бухгалтерского учета ФСБУ 6/2020 "Основные средства" и ФСБУ 26/2020 "Капитальные вложения"

Основная литература:

4. "Python для анализа данных" — Уэсли Чан (Wes McKinney), 2022
5. "Изучаем Python" — Марк Лутц (Mark Lutz), 2021
6. "Flask Web Development" — Мигель Гринберг (Miguel Grinberg), 2022

Интернет-ресурсы:

7. Телеграмм бот на Python [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/442800/ (дата обращения 05.05.2025)
8. telebot быстро и понятно [Электронный ресурс]. Режим доступа: https://habr.com/ru/articles/580408/ (дата обращения 04.05.2025)
9. Размещаем бота для телеграмм [Электронный ресурс]. Режим доступа: https://timeweb.com/ru/community/articles/razmeshchaem-bota-dlya-telegram-ot-vyborahostinga-do-zapuska (дата обращения 01.04.2025)
10. Графический клиент MySQL Workbench [Электронный ресурс]. Режим доступа: https://metanit.com/sql/mysql/1.3.php (дата обращения 21.04.2025)
11. MySQL Workbench: установка и настройка для работы с базами данных [Электронный ресурс]. Режим доступа: https://selectel.ru/blog/tutorials/mysql-workbench-installation/ (дата обращения 10.05.2025)

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

API - Application Programming Interface (Интерфейс программирования приложений)

IDE - Integrated Development Environment (Интегрированная среда разработки)

JSON - JavaScript Object Notation (Формат обмена данными)

LibreOffice - Офисный пакет с открытым исходным кодом

MySQL - My Structured Query Language (Система управления базами данных)

NodeJS - Node.js (Среда выполнения JavaScript)

PM2 - Process Manager 2 (Менеджер процессов)

PyCharm - Интегрированная среда разработки для Python

SQL - Structured Query Language (Язык структурированных запросов)

SFTP - Secure File Transfer Protocol (Протокол безопасной передачи файлов)

SSH - Secure Shell (Протокол безопасного соединения)

Termius - Клиент для SSH и SFTP

Timeweb - Хостинг-провайдер

URL - Uniform Resource Locator (Унифицированный указатель ресурса)

VDS - Virtual Dedicated Server (Виртуальный выделенный сервер)

VPS - Virtual Private Server (Виртуальный частный сервер)

XML - eXtensible Markup Language (Расширяемый язык разметки)

Приложение А: Исходные коды проекта

А.1 Файл main.py

```
import os
import file_Markups
from file_Bot import *
import file_Create_ad
import file_my_ads
import file_find_ads
import file_Moderation
import file_menu
from file_mysql import *
@bot.message_handler(commands=['start'])
def start(message):
    cursor.execute(f"SELECT COUNT(1) FROM BANNED_USERS WHERE
user_id=%s", (message.from_user.id,))
    check_ban = cursor.fetchall()
    for i in check_ban:
        if str(i) == "(0,)":
            cursor.execute(f"SELECT COUNT(1) FROM ID_USER_TABLE WHERE
user_id=%s", (message.from_user.id,))
            check_id = cursor.fetchall()
            for i in check_id:
                if str(i) == "(0,)":
                    bot.send_message(message.chat.id, '   Отказ от
ответственности.\n\nСоздатель бота не несет ответственности за украденные данные
пользователя, за распространение информации или любые другие действия,
совершенные пользователями с использованием данного бота. Использование бота
осуществляется на свой страх и риск. Владелец бота не несет ответственности за
какие-либо убытки, ущерб, проблемы или последствия, возникшие в результате
```

использования данного бота или связанные с ним. Пользователь самостоятельно несет ответственность за свои действия.',
reply_markup=file_Markups.markup_no_responsibility)
            else:
                bot.send_message(message.chat.id, f"Добро пожаловать, {message.from_user.first_name}!", reply_markup=file_Markups.markup_menu)
        else:
            bot.send_message(message.chat.id, "Вы забанены!")
@bot.message_handler(content_types=['text'])
def text(message):
        cursor.execute(f'SELECT COUNT(1) FROM BANNED_USERS WHERE user_id=%s", (message.from_user.id,))
        check_ban = cursor.fetchall()
        for i in check_ban:
            if str(i) == "(0,)":
                if message.text == "Я согласен":
                    bot.send_sticker(message.chat.id, "CAACAgIAAxkBAAEMTkFnpRwt_4UIYq_g3nCPhGZkkc4zRwACBQADwDZPE_lqX5qCa011NgQ")
                    bot.send_message(message.chat.id, f"Добро пожаловать, {message.from_user.first_name}!", reply_markup=file_Markups.markup_menu)
                    cursor.execute(f'INSERT INTO ID_USER_TABLE(user_id) VALUES (%s)", (message.from_user.id,))
                    cursor.execute(f'INSERT INTO criteries(id) VALUES ({message.from_user.id})")
                elif message.text == "Создать объявление":
                    file_Create_ad.start_create_ad(message)
                elif 'Запросы ads' in message.text:
                    file_Moderation.Moderation(message)
                elif message.text == "Найти объявления":

```
            file_find_ads.func_start(message)
        elif message.text == "Моё объявление":
            file_my_ads.my_ads(message)
        elif message.text == "Меню":
            file_menu.func_menu1(message)
        elif message.text == "Удалить":
            try:
                cursor.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s",
(message.from_user.id,))
                a = cursor.fetchall()
                for i in a:
                    try:
                        os.remove(i[7])
                    except:
                        pass
                cursor.execute(f"DELETE FROM ADS_TABLE WHERE user_id=%s",
(message.from_user.id,))
                bot.send_message(message.chat.id, "Ваше объявление удалено!",
reply_markup=file_Markups.markup_go_menu)
            except Exception as err:
        elif message.text == "Все объявления":
            try:
                cursor.execute(f"SELECT COUNT(1) from ADS_TABLE")
                a123 = cursor.fetchall()
                for i in a123:
                    if str(i) == "(0,)":
                        bot.send_message(message.chat.id, "Объявлений нет")
                    else:
                        cursor.execute(f"SELECT * FROM ADS_TABLE")
                        a = cursor.fetchall()
```

```
                    for i in a:
                        if i[3] == None:
                            bot.send_message(message.chat.id,
                            f"- Имя: {i[1]}\n"
                            f"- Возраст: {i[6]}\n"
                            f"- Страна: {i[7]}\n"
                            f"- Описание: {i[2]}\n"
                            f"- Назвние игры: {i[8]}\n"
                            f"- Статус: {i[4]}")
                        else:
                            photo = open(f"{i[3]}", "rb")
                            bot.send_photo(message.chat.id, photo=photo, caption=
                            f"- Имя: {i[1]}\n"
                            f"- Возраст: {i[6]}\n"
                            f"- Страна: {i[7]}\n"
                            f"- Описание: {i[2]}\n"
                            f"- Назвние игры: {i[8]}\n"
                            f"- Статус: {i[4]}")
                except Exception as err: pass
            else:
                bot.send_message(message.chat.id, "Простите, я не понимаю вас",
reply_markup=file_Markups.markup_menu)
        else:
            bot.send_message(message.chat.id, "Вы забанены!")
bot.polling(none_stop=True, interval=0)
```

A.2 Файл file_my_ads.py

```
import file_Bot
import mysql.connector
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
def my_ads(message):
    try:
        cursor.execute(f"SELECT COUNT(1) from ADS_TABLE WHERE user_id=%s",
(message.from_user.id,))
        b = cursor.fetchall()
        for i in b:
            if str(i) == "(0,)":
                bot.send_message(message.chat.id, "У вас нет активных объявлений, вы
можете создать!", reply_markup=file_Markups.markup_menu)
            else:
                cursor.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s",
(message.from_user.id,))
                a = cursor.fetchall()
                for i in a:
                    if i[3] == None:
                        bot.send_message(message.chat.id, f"- Имя: {i[1]}\n"
                                                          f"- Возраст: {i[6]}\n"
                                                          f"- Страна: {i[7]}\n"
                                                          f"- Описание: {i[2]}\n"
                                                          f"- Теги: {i[8]}\n"
```

```
                                f"- Статус: {i[4]}",
reply_markup=file_Markups.markup_delete_ad)
            else:
                photo = open(f"{i[3]}", "rb")
                bot.send_photo(message.chat.id, photo=photo, caption=
                                f"- Имя: {i[1]}\n"
                                f"- Возраст: {i[6]}\n"
                                f"- Страна: {i[7]}\n"
                                f"- Описание: {i[2]}\n"
                                f"- Теги: {i[8]}\n"
                                f"- Статус: {i[4]}",
reply_markup=file_Markups.markup_delete_ad)
    except Exception as err:
        print(err)
```

A.3 Файл file_Create_ad.py

```
import mysql.connector
import file_Bot
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
def start_create_ad(message):
    cursor.execute(f"SELECT COUNT(1) from ADS_TABLE WHERE user_id=%s",
(message.from_user.id,))
    t1 = cursor.fetchall()
    for i in t1:
        if str(i) == "(0,)":
            try:
                cursor.execute(f"INSERT INTO ADS_TABLE (user_id, moderation) VALUES
(%s, %s)",  (message.from_user.id, 'На рассмотрении ◔◔'))
                msg = bot.send_message(message.chat.id, "Хотите добавить фото к
объявлению?",
                                       reply_markup=file_Markups.markup_photo)
                bot.register_next_step_handler(msg, func_send_photo_or_not)
            except Exception as err:
                print(err)
        else:
            bot.send_message(message.chat.id, "Вы уже создали объявление!",
reply_markup=file_Markups.markup_go_menu)
def func_send_photo_or_not(message):
    if message.text == "Да":
```

```
        msg = bot.send_message(message.chat.id, "Отправьте фото...",
reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, func_photo)
    elif message.text == "Нет":
        msg = bot.send_message(message.chat.id, "Как вас зовут?",
reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, func_name)
def func_photo(message):
    try:
        photo_id = message.photo[-1].file_id
        photo_file = bot.get_file(photo_id)
        download_file = bot.download_file(photo_file.file_path)
        src = "photos/" + photo_file.file_path.replace('photos/', '')
        with open(src, 'wb') as new_file:
            new_file.write(download_file)
        cursor.execute(f"UPDATE ADS_TABLE SET photo_way=%s WHERE
user_id=%s", (f"{str(src)}", message.from_user.id,))
        db.commit()
        msg = bot.send_message(message.chat.id, f"Как вас зовут?",
reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, func_name)
    except:
        msg = bot.send_message(message.chat.id, "Пожалуйста, отправьте фотографию!")
        bot.register_next_step_handler(msg, func_photo)
def func_name(message):
    cursor.execute(f"UPDATE ADS_TABLE SET name=%s WHERE user_id=%s",
(f"{message.text}", message.from_user.id,))
    db.commit()
    msg = bot.send_message(message.chat.id, "Укажите свой возраст")
    bot.register_next_step_handler(msg, func_age)
```

```
def func_age(message):
    age = message.text
    if not age.isdigit():
        msg = bot.send_message(message.chat.id, "Пожалуйста, введите число!")
        bot.register_next_step_handler(msg, func_age)
    elif int(age) > 60:
        msg = bot.send_message(message.chat.id, "Пожалуйста, введите свой настоящий возраст!")
        bot.register_next_step_handler(msg, func_age)
    elif int(age) < 10:
        msg = bot.send_message(message.chat.id, "Пожалуйста, введите свой настоящий возраст!")
        bot.register_next_step_handler(msg, func_age)
    else:
        cursor.execute(f"UPDATE ADS_TABLE SET age=%s WHERE user_id=%s", (int(message.text), message.from_user.id,))
        db.commit()
        msg = bot.send_message(message.chat.id, "Введите текст к своему объявлению")
        bot.register_next_step_handler(msg, func_game)
def func_game(message):
    cursor.execute(f"UPDATE ADS_TABLE SET text=%s WHERE user_id=%s", (f"{message.text}", message.from_user.id,))
    db.commit()
    msg = bot.send_message(message.chat.id, "Введите название игры")
    bot.register_next_step_handler(msg, func_text)
def func_text(message):
    cursor.execute(f"UPDATE ADS_TABLE SET game=%s WHERE user_id=%s", (f"{message.text.lower()}", message.from_user.id,))
    db.commit()
```

```
    msg = bot.send_message(message.chat.id, "Из какой вы страны?",
reply_markup=file_Markups.markup_country)
    bot.register_next_step_handler(msg, func_final)
def func_final(message):
    cursor.execute(f"UPDATE ADS_TABLE SET country=%s WHERE user_id=%s",
(f"{message.text.lower()}", message.from_user.id,))
    db.commit()
    cursor.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s",
(message.from_user.id,))
    a = cursor.fetchall()
    for i in a:
        if i[3] == None:
            msg = bot.send_message(message.chat.id, f"- Имя: {i[1]}\n"
                                                    f"- Возраст: {i[6]}\n"
                                                    f"- Страна: {i[7]}\n"
                                                    f"- Описание: {i[2]}\n"
                                                    f"- Игра: {i[8]}",
reply_markup=file_Markups.markup_right_wrong)
            bot.register_next_step_handler(msg, func_correct_wrong)
        else:
            photo = open(f"{i[3]}", "rb")
            msg2 = bot.send_photo(message.chat.id, photo=photo, caption=
                                                    f"- Имя: {i[1]}\n"
                                                    f"- Возраст: {i[6]}\n"
                                                    f"- Страна: {i[7]}\n"
                                                    f"- Описание: {i[2]}\n"
                                                    f"- Игра: {i[8]}",
reply_markup=file_Markups.markup_right_wrong)
            bot.register_next_step_handler(msg2, func_correct_wrong)
def func_correct_wrong(message):
```

```
    if message.text == "Верно":
        bot.send_message(message.chat.id, "Ваше объявление отправлено на модерацию...", reply_markup=file_Markups.markup_go_menu)
    elif message.text == "Исправить":
        msg = bot.send_message(message.chat.id, "Начнем с начала...\nКак вас зовут?", reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, func_name)
```

A.4 Файл file_Moderation.py

```
import file_Bot
import mysql.connector
import traceback
import file_Markups
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor2 = db.cursor()
cursor2 = db.cursor(buffered=True)
def Moderation(message):
    try:
        cursor.execute("SET AUTOCOMMIT = 1")
        cursor.execute("SELECT COUNT(1) FROM ADS_TABLE WHERE
moderation=%s", ('На рассмотрении ',))
        find_ads = cursor.fetchall()
        for i in find_ads:
            if str(i) == "(0,)":
                bot.send_message(message.chat.id, 'Объявлений не найдено')
            else:
                cursor.execute("SELECT * FROM ADS_TABLE WHERE moderation='На
рассмотрении '")
                row = cursor.fetchone()
                search(message, row)
    except Exception as err: pass
def ads(message, cursor):
    row = cursor.fetchone()
    if row:
```

```
        search(message, row)
    else:
        bot.send_message(message.chat.id, 'Больше объявлений нет',
reply_markup=file_Markups.markup_go_menu)
def search(message, row):
    try:
        if row[3] == None:
            msg = bot.send_message(message.chat.id, f"- Имя: {row[1]}\n"
                                                    f"- Возраст: {row[6]}\n"
                                                    f"- Страна: {row[7]}\n"
                                                    f"- Описание: {row[2]}\n"
                                                    f"- Игра: {row[8]}",
                                   reply_markup=file_Markups.markup_moderation)
            bot.register_next_step_handler(msg, solution, row[0], cursor)
        else:
            photo = open(f"{row[3]}", "rb")
            msg = bot.send_photo(message.chat.id, photo=photo, caption=
                                                     f"- Имя: {row[1]}\n"
                                                     f"- Возраст: {row[6]}\n"
                                                     f"- Страна: {row[7]}\n"
                                                     f"- Описание: {row[2]}\n"
                                                     f"- Игра: {row[8]}",
                                 reply_markup=file_Markups.markup_moderation)
            bot.register_next_step_handler(msg, solution, row[0], cursor)
    except Exception as e:
        bot.send_message(message.chat.id, "Что-то пошло не так",
reply_markup=file_Markups.markup_menu)
def solution(message, id, cursor):
    if message.text == "Принять":
        try:
```

```
            cursor2.execute(f"UPDATE ADS_TABLE SET moderation=%s WHERE
user_id=%s", ("Активно ", int(id),))
            bot.send_message(message.chat.id, "Объявление принято!")
            bot.send_message(id, "Поздравляем! Ваше объявление прошло проверку!")
            ads(message, cursor)
        except Exception as err:
    elif message.text == "Отклонить":
        msg = bot.send_message(message.chat.id, "Причина недопуска объявления:...",
reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, non_admission, id, cursor)
    elif message.text == "Бан":
        try:
            bot.send_message(id, "Вы были забанены!")
            bot.send_message(message.chat.id, "Был выдан бан")
            cursor2.execute(f"INSERT INTO BANNED_USERS (user_id) VALUES (id)")
            cursor2.execute(f"DELETE FROM ADS_TABLE WHERE user_id=%s", (id,))
            cursor2.execute(f"DELETE FROM ID_USER_TABLE WHERE user_id(id)")
ads(message, cursor)
        except Exception as err:
def non_admission(message, id, cursor):
    try:
        cursor2.execute(f"UPDATE ADS_TABLE SET moderation=%s WHERE
user_id=%s", (f"Отклонено по причине: {message.text}", int(id),))
        cursor2.execute(f"SELECT * FROM ADS_TABLE WHERE user_id=%s", (int(id),))
        bot.send_message(id, f" Ваше объявление было недопущено по причине:
{message.text}")
        ads(message, cursor)
    except Exception as err:
        print(err)
```

A.5 Файл file_find_ads.py

```
import mysql.connector
import file_Bot
import file_Markups
import traceback
import file_menu
bot = file_Bot.bot
db = mysql.connector.connect(host="localhost", user="root", password="admin",
database="botforsearchingonlinefriends")
cursor = db.cursor()
cursor = db.cursor(buffered=True)
cursor.execute("SET AUTOCOMMIT = 1")
def func_start(message):
    cursor.execute(f"UPDATE criteries SET game=Null, country=Null, age=Null WHERE
id={message.from_user.id}")
    msg = bot.send_message(message.chat.id, "Введите название игры",
reply_markup=file_Markups.delete_markup)
    bot.register_next_step_handler(msg, func_game)
def func_game(message):
    cursor.execute(f"UPDATE criteries SET game='{message.text.lower()}'")
    msg = bot.send_message(message.chat.id, "Введите минимальный возраст игрока",
reply_markup=file_Markups.markup_ages)
    bot.register_next_step_handler(msg, func_age)
def func_age(message):
    age = message.text
    if not age.isdigit():
        msg = bot.send_message(message.chat.id, "Пожалуйста, выберите число из
предоставленной клавиатуры или укажите самостоятельно!",
reply_markup=file_Markups.markup_ages)
        bot.register_next_step_handler(msg, func_age)
```

```
    else:
        cursor.execute(f"UPDATE criteries SET age={int(message.text)}")
        msg = bot.send_message(message.chat.id, "Из какой страны искать
объявления?\n(Пожалуйста, пишите без ошибок!)",
reply_markup=file_Markups.markup_country_and_no_matter)
        bot.register_next_step_handler(msg, func_country)
def func_country(message):
    try:
        cursor.execute(f"UPDATE criteries SET country='{message.text.lower()}'")
        cursor.execute(f"SELECT * FROM criteries WHERE id={message.from_user.id}")
        data = cursor.fetchall()
        if message.text == "Не имеет значения":
            cursor.execute(
                f"SELECT COUNT(1) FROM ADS_TABLE WHERE age>{data[0][3] + 1}
AND moderation='Активно ✓' AND game LIKE '%{data[0][1]}%' OR text LIKE
'%{data[0][1]}%'")
            a = cursor.fetchall()
            for kol_vo_ads in a:
                if str(kol_vo_ads) == "(0,)":
                    bot.send_message(message.chat.id, "К сожалению, ничего не найдено по
вашим критериям :(",
                                     reply_markup=file_Markups.markup_menu)
                else: cursor.execute( f"SELECT * FROM ADS_TABLE WHERE
            age>{data[0][3] + 1} AND moderation='Активно ✓' AND game LIKE
            '%{data[0][1]}%' OR text LIKE '%{data[0][1]}%'")
                    row = cursor.fetchone()
                    search(message, row)
        else:
```

```
            cursor.execute(f"SELECT COUNT(1) FROM ADS_TABLE WHERE
country='{data[0][2]}' AND age>{data[0][3] + 1} AND moderation='Активно ✅' AND
game LIKE '%{data[0][1]}%' OR text LIKE '%{data[0][1]}%'")
            a = cursor.fetchall()
            for kol_vo_ads in a:
                if str(kol_vo_ads) == "(0,)":
                    bot.send_message(message.chat.id, "К сожалению, ничего не найдено по
вашим критериям :(", reply_markup=file_Markups.markup_go_menu)
                else: cursor.execute(f"SELECT * FROM ADS_TABLE WHERE
country='{data[0][2]}' AND age>{data[0][3] + 1} AND moderation='Активно ✅' AND
game LIKE '%{data[0][1]}%' OR text LIKE '%{data[0][1]}%'")
                    row = cursor.fetchone()
                    search(message, row)
    except Exception as err: pass
def ads(message, cursor):
    row = cursor.fetchone()
    if row:
        search(message, row)
    else: bot.send_message(message.chat.id, 'Больше объявлений нет :(',
reply_markup=file_Markups.markup_go_menu)
def like_or_not_like(message, id, cursor):
    if message.text == "Отправить сообщение":
        msg = bot.send_message(message.chat.id, f"Какое сообщение отправить?",
reply_markup=file_Markups.delete_markup)
        bot.register_next_step_handler(msg, send_message_to_user, id, cursor)
    elif message.text == "Следующий":
        ads(message, cursor)
    elif message.text == "Меню":
        file_menu.func_menu1(message)
```

```
    else:
        bot.send_message(message.chat.id, "Пожалуйста, пользуйтесь предоставленной клавиатурой!", reply_markup=file_Markups.markup_menu)
def send_message_to_user(message, id, cursor):
    bot.send_message(id, f'вам пришло сообщение от @{message.from_user.username}\nСодержание: {message.text}')
    ads(message, cursor)
def search(message, row):
    try:
        if row[3] == None:
            msg = bot.send_message(message.chat.id, f"- Имя: {row[1]}\n"
                                                    f"- Возраст: {row[6]}\n"
                                                    f"- Игра: {row[8]}\n"
                                                    f"- Страна: {row[7]}\n"
                                                    f"- Описание: {row[2]}\n",
                                   reply_markup=file_Markups.markup_choose)
            bot.register_next_step_handler(msg, like_or_not_like, row[0], cursor)
        else:
            photo = open(f"{row[3]}", "rb")
            msg = bot.send_photo(message.chat.id, photo=photo, caption=
                                                    f"- Имя: {row[1]}\n"
                                                    f"- Возраст: {row[6]}\n"
                                                    f"- Игра: {row[8]}\n"
                                                    f"- Страна: {row[7]}\n"
                                                    f"- Описание: {row[2]}\n",
                                 reply_markup=file_Markups.markup_choose)
            bot.register_next_step_handler(msg, like_or_not_like, row[0], cursor)
    except Exception as e:
        bot.send_message(message.chat.id, "Что-то пошло не так... :(", reply_markup=file_Markups.markup_menu)
```

A.6 Файл file_Markups.py

```
from telebot import types
markup_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_create = types.KeyboardButton("Создать объявление")
btn_find = types.KeyboardButton("Найти объявления")
btn_my_ad = types.KeyboardButton("Моё объявление")
markup_menu.add(btn_create, btn_find, btn_my_ad)
markup_go_menu = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_menu = types.KeyboardButton('Меню')
markup_go_menu.add(btn_menu)
markup_photo = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_yes = types.KeyboardButton("Да")
btn_no = types.KeyboardButton("Нет")
markup_photo.add(btn_yes, btn_no)
markup_right_wrong = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_right = types.KeyboardButton("Верно")
btn_wrong = types.KeyboardButton("Исправить")
markup_right_wrong.add(btn_right, btn_wrong)
markup_delete_ad = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_delete_ad = types.KeyboardButton("Удалить")
markup_delete_ad.add(btn_delete_ad, btn_menu)
markup_ages = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_13 = types.KeyboardButton("13")
btn_16 = types.KeyboardButton("16")
btn_18 = types.KeyboardButton("18")
btn_20 = types.KeyboardButton("20")
btn_25 = types.KeyboardButton("25")
markup_ages.add(btn_13, btn_16, btn_18, btn_20, btn_25)
markup_moderation = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_accept = types.KeyboardButton("Принять")
```

```
btn_reject = types.KeyboardButton("Отклонить")
btn_ban = types.KeyboardButton('Бан')
markup_moderation.add(btn_accept, btn_reject, btn_ban)
markup_admin_panel = types.ReplyKeyboardMarkup(resize_keyboard=True)
text = "Запросы ads"
btn_modering_ads = types.KeyboardButton(text)
btn_admin_show_all_ads = types.KeyboardButton("Все объявления")
markup_admin_panel.add(btn_create, btn_find, btn_my_ad, btn_modering_ads,
btn_admin_show_all_ads)
markup_choose = types.ReplyKeyboardMarkup(resize_keyboard=True)
choose1 = types.KeyboardButton("Отправить сообщение")
choose2 = types.KeyboardButton("Следующий")
markup_choose.add(choose1, btn_menu,choose2)
markup_no_responsibility = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_no_responsibility = types.KeyboardButton("Я согласен")
markup_no_responsibility.add(btn_no_responsibility)
markup_country_and_no_matter = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_rus = types.KeyboardButton("Россия")
btn_belrus = types.KeyboardButton("Беларусь")
btn_no_matter = types.KeyboardButton("Не имеет значения")
markup_country_and_no_matter.add(btn_belrus, btn_rus, btn_no_matter)
markup_country = types.ReplyKeyboardMarkup(resize_keyboard=True)
btn_rus = types.KeyboardButton("Россия")
btn_belrus = types.KeyboardButton("Беларусь")
markup_country.add(btn_belrus, btn_rus)
delete_markup = types.ReplyKeyboardRemove()
```

A.7 Файл file_Bot.py

```
import telebot
TOKEN = "7548926182:AAHujrriBEI2iymJAsV3ZkCMNx06xJKuHLU"
bot = telebot.TeleBot(TOKEN)
```