ФЕДЕРАЛЬНОЕ АГЕНТСТВО МОРСКОГО И РЕЧНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Государственный университет морского и речного флота имени адмирала С.О. Макарова»
(ФГБОУ ВО «ГУМРФ имени адмирала С.О. Макарова»)
Воронежский филиал

Кафедра математики, информационных систем и технологий
Направление подготовки 09.03.02 Информационные системы и технологии
(код, наименование направления подготовки/специальности)
Форма обучения очная ускоренная

«К ЗАЩИТЕ ДОПУЩЕН(А)»
Заведующий кафедрой
_____
(подпись)
Черняева С. Н.
(ФИО)
_____ 20___

Выпускная квалификационная работа

Обучающегося Седько Тимофея Геннадьевича
(фамилия, имя, отчество)
Вид работы выпускная квалификационная работа бакалавра
(выпускная квалификационная работа бакалавра, специалиста, магистра)

Пояснительная записка

Тема Разработка модуля «Новостная лента» информационной системы управления корпоративными коммуникациями (на примере АО ГК «ЭФКО»)
(полное название темы квалификационной работы, в соответствии с приказом об утверждении тематики ВКР)

Руководитель работы доцент Черняева С.Н.
(должность, подпись, фамилия, инициалы, дата)
Консультант _____
(при наличии) (должность, подпись, фамилия, инициалы, дата)
Консультант _____
(должность, подпись, фамилия, инициалы, дата)
Обучающийся Седько Т.Г.
(подпись, фамилия, инициалы, дата)

Воронеж
2024

ФЕДЕРАЛЬНОЕ АГЕНТСТВО МОРСКОГО И РЕЧНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Государственный университет морского и речного флота имени адмирала С.О. Макарова»
(ФГБОУ ВО «ГУМРФ имени адмирала С.О. Макарова»)
Воронежский филиал

Кафедра математики, информационных систем и технологий
Направление подготовки 09.03.02 Информационные системы и технологии
(код, наименование направления подготовки/специальности)
Форма обучения очная ускоренная

УТВЕРЖДАЮ
Заведующий кафедрой
_____
(подпись)
Черняева С. Н.
(ФИО)
_____ 2024

Задание
на выпускную квалификационную работу

Вид работы ВКР бакалавра
(ВКР бакалавра, ВКР специалиста, ВКР магистра)

Обучающемуся Седько Тимофею Геннадьевичу
(фамилия, имя, отчество)

Тема Разработка модуля «Новостная лента» информационной системы управления корпоративными коммуникациями (на примере АО ГК «ЭФКО»)

Утверждена приказом ректора университета от _____ 2024, № _____

Срок сдачи законченной работы _____ 20___

Исходные данные (или цель ВКР):
Разработать модуль «Новостная лента» для управления корпоративными коммуникациями

Перечень подлежащих исследованию, разработке, проектированию вопросов (краткое содержание ВКР):
(актуальность темы, цели и задачи ВКР; аналитический обзор литературных источников; постановка задачи исследования, разработки, проектирования; содержание процедуры исследования, разработки, проектирования; обсуждение результатов; дополнительные вопросы, подлежащие разработке; заключение – выводы по работе в целом, оценка степени решения поставленных задач, практические рекомендации; и др.)

– Введение. Актуальность выбранной темы, цель и задачи ВКР

(наименование вопроса, раздела и его краткое содержание)

– Исследовательский раздел. Анализ, выбор, обзор аналогов, характеристика предприятия, постановка задачи, экономическая сущность и формализация расчетов подзадач

(наименование вопроса, раздела и его краткое содержание)

– Проектный раздел. Информационное, программное, технологическое обеспечение задачи и руководство пользователя

(наименование вопроса, раздела и его краткое содержание)

– Заключение. Выводы по работе в целом. Оценка степени решения поставленных задач

(наименование вопроса, раздела и его краткое содержание)

Практические рекомендации

Перечень графического материала (или презентационного материала):

1. Титульный лист
2. Цель и задачи ВКР
3. Актуальность темы
4. Анализ современных систем
5. Анализ современных систем
6. Выбор системы
7. Проектирование решения
8. Модуль "Парсер новостей"
9. Модуль "Чат-бот для управления тикетами"
10. Модуль "Бот для предложений новостей"
11. Экономическая эффективность
12. Результаты ВКР

Консультанты по разделам ВКР (при наличии):

1. _____

(наименование раздела, ученая степень, ученое звание и должность, ФИО консультанта)

2. _____

(наименование раздела, ученая степень, ученое звание и должность, ФИО консультанта)

3. _____

(наименование раздела, ученая степень, ученое звание и должность, ФИО консультанта)

Дата выдачи задания: _____ 20___

Задание согласовано и принято к исполнению: _____ 20___

Руководитель ВКР: _____ _____

(должность, ученая степень, ученое звание, ФИО) (подпись)

Обучающийся: ИТ-3-2 Седько Тимофей Геннадьевич _____

(учебная группа, ФИО)

СОДЕРЖАНИЕ

Введение 6

1. Исследовательский раздел 8

1.1. Анализ современных систем управления корпоративными коммуникациями 8

1.2. Выбор системы управления корпоративными коммуникациями для реализации новостной ленты 21

1.3. Обзор аналогов новостных лент в системах управления корпоративными коммуникациями 29

1.4. Характеристика АО «УК ЭФКО» 34

1.5. Структура «ЭФКО» 35

1.6. Постановка задачи 37

1.7. Экономическая сущность задачи 41

1.8. Формализация расчетов подзадач 43

2. Проектный раздел 47

2.1. Информационное обеспечение задачи 47

2.2. Программное обеспечение задачи 52

2.3. Технологическое обеспечение задачи 60

2.4. Руководство пользователя 65

Заключение 75

Список использованных источников 77

Приложение А 81

Приложение Б 82

Приложение В 83

Приложение Г 84

Приложение Д 89

Приложение Е..... 96

ВВЕДЕНИЕ

Актуальность выбранной темы обусловлена рядом причин. Во-первых, без слаженной работы и общих ценностей невозможно построить эффективный бизнес. Во-вторых, с течением времени внутренние коммуникации становятся все более значимым инструментом в работе бизнесов, для своевременного обмена информацией. В-третьих, внутренние коммуникации должны помогать работнику понимать корпоративную культуру, цели и ценности. Сотрудники должны быть в курсе событий и решений, которые касаются работы всех отделов. При этом нельзя забывать, что средства и методы связи должны быть максимально удобны для сотрудников.

Целью исследования является, улучшение системы оповещения сотрудников, посредством разработки модуля «Новостная лента» для управления корпоративными коммуникациями. Задачами исследования является:

– анализ существующих решений и требований к функциональности модуля;
– обоснование выбора основных проектируемых решений;
– разработка модуля «Новостная лента»;
– тестирование и отладка разработанного модуля;
– обоснование экономической эффективности работы;
– оценка эффективности модуля.

Объектом исследования, будет являться мессенджеры как системы управления корпоративными коммуникациями. Предметом исследования будет являться модуль «Новостная лента».

В качестве методов исследования, мы будем использовать контент-анализ, чтобы анализировать содержания текстовых и мультимедийные данные для выявления значимых тенденций и закономерностей. Для оценки потенциала внедрения новостного модуля, проанализируем функционал,

преимущества и недостатки программ. Также проанализируем распространённость и охват аудитории этих программ.

Работа включает в себя, исследовательский раздел, в котором проводиться анализ современных систем управления корпоративными коммуникациями, выбор современных систем управления корпоративными коммуникациями для реализации новостной ленты и обзор аналогов новостных лент в современных систем управления корпоративными коммуникациями, характеристика и структура предприятия, расчёт экономической сущности задачи и формализация расчетов подзадач.

Проектный раздел с информационным обеспечением задачи, описанием информационной модели, перечислением используемых классификаторов и системы кодирования, а также с характеристика первичных документов с нормативно-справочной и входной оперативной информацией и баз данных.

Программное обеспечение задачи, входящее в проектный раздел, включает описание логики программ, а также модулей, технологическим обеспечением задачи и руководство пользователя.

1. ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

1.1. Анализ современных систем управления корпоративными коммуникациями

Проанализируем современные систем управления корпоративными коммуникациями, имеющие ключевое значение в современных организациях. Они обеспечивают эффективное взаимодействие между сотрудниками и разными отделами, «Стратегические функции и архитектура корпоративных коммуникаций организации. Формальные коммуникации: нисходящие и восходящие; горизонтальные коммуникации (между подразделениями одного уровня в иерархии структуры).» [1]. Вот несколько современных подходов и систем, которые используются или могут использоваться для управления корпоративными коммуникациями.

1.1.1. Мессенджеры и чаты

«В наше время мессенджеры стали неотъемлемой частью повседневной жизни человека, и ежедневная переписка превратилась в настолько привычное дело для нас, что мы не можем представить свою жизнь без неё.» [3].

Telegram, рисунок 1, многофункциональный мессенджер, доступный на разных устройствах. Поддерживающий голосовые вызовы через интернет и предлагает большой спектр инструментов для общения. Пользователи могут делиться голосовыми сообщениями, видео, изображениями, стикерами и отправлять файлы любого размера. Мессенджер можно настроить под индивидуальные нужды пользователей.

Преимущества:

– высокая скорость работы;

– возможность запустить рекламу через Telegram Ads или посев у блогеров, чтобы набрать аудиторию в чат – бота, не уводя пользователя из мессенджера;

– расширенные возможности безопасности (закрытые чаты, сквозное шифрование, самоуничтожающиеся сообщения);

– синхронизация чатов на разных устройствах.

Недостатки:

– технические проблемы в Android – версии;

– некоторые функции теперь доступны только premium – аккаунтам за дополнительную плату;

– проблемы с фильтрацией спама и рекламы.

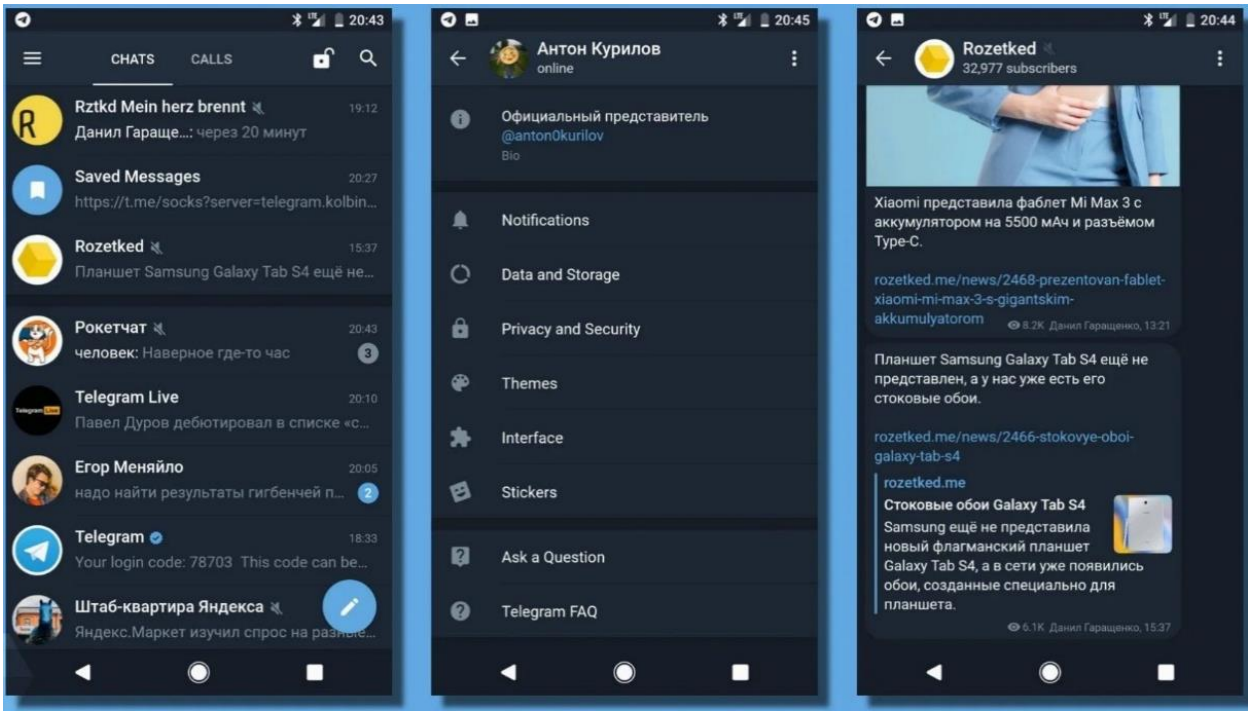

Рисунок 1 – Интерфейс мессенджера Telegram

WhatsApp, рисунок 2, принадлежит Meta[1], остается одним из самых популярных мессенджеров в России. Он привлекает внимание инвесторов, хотя нововведения в последнее время появляются не так часто. WhatsApp часто заимствует функции у конкурентов, но он все еще популярен среди бизнеса, фрилансеров и тех, кто работает с клиентами.

Преимущества:

[1] Деятельность Meta (и социальных сетей Facebook и Instagram) запрещена на территории Российской Федерации на основании Федерального закона от 25.07.2002 № 114-ФЗ «О противодействии экстремистской деятельности».

– отдельная версия для бизнеса;

– автоматическое сохранение сообщений в офлайн-режиме;

– обмен гео – метками и удобные виджеты.

Недостатки:

– отсутствие настоящей кроссплатформенности, требует наличия телефона для использования на ПК;

– устаревший дизайн интерфейса;

– технические неполадки после обновлений.

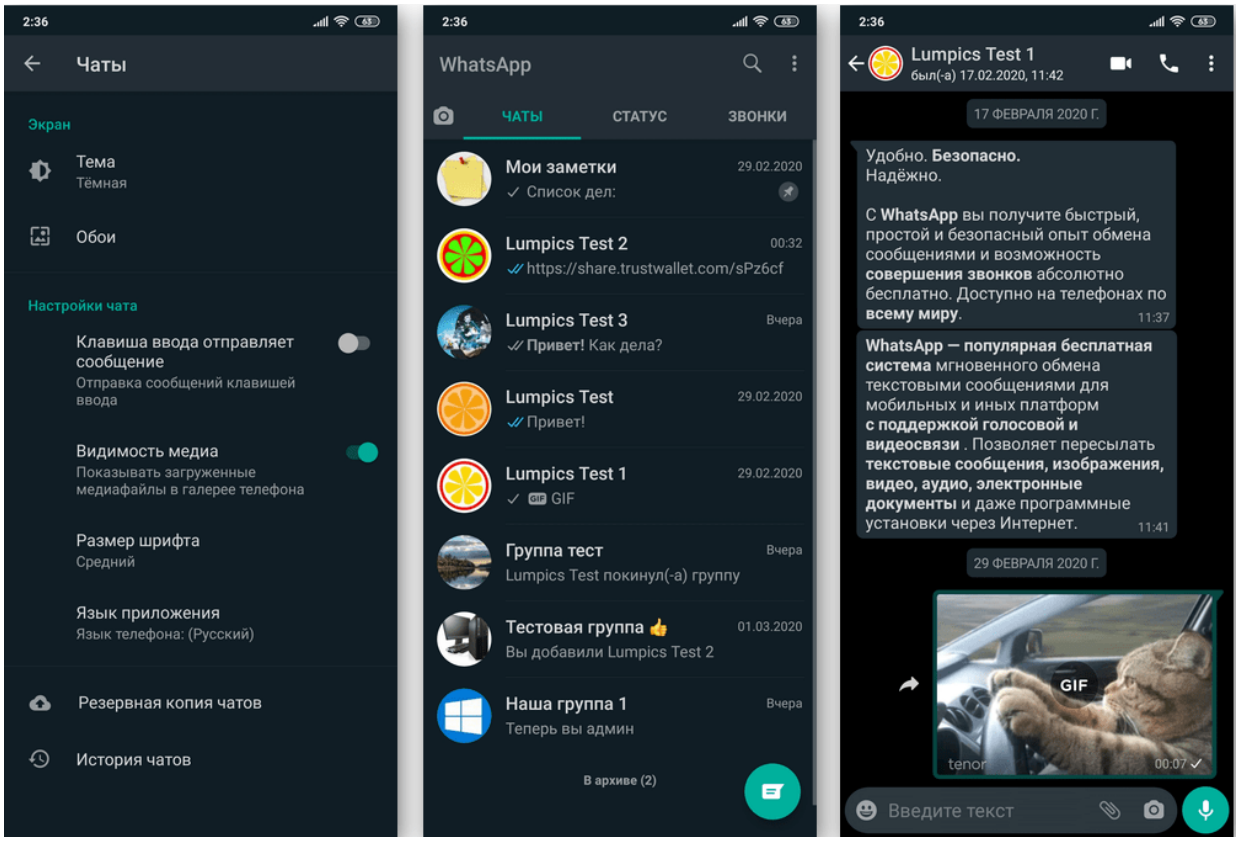

Рисунок 2 – Интерфейс мессенджера WhatsApp

Viber предлагает базовые функции для общения, такие как голосовые и видеозвонки, текстовые сообщения и обмен файлами. Хотя он не предлагает много уникальных инноваций, Viber фокусируется на безопасности и производительности для обеспечения быстрого и надежного общения.

Преимущества:

– поддержка разных платформ, включая смарт-часы;

– встроенная камера с AR – эффектами и фильтрами;

– гибкая настройка сообществ и групп.

Недостатки:

– проблемы с занимаемым местом в памяти устройств;

– неудобное взаимодействие с медиатекой;

– технические сбои и необходимость повторной авторизации.

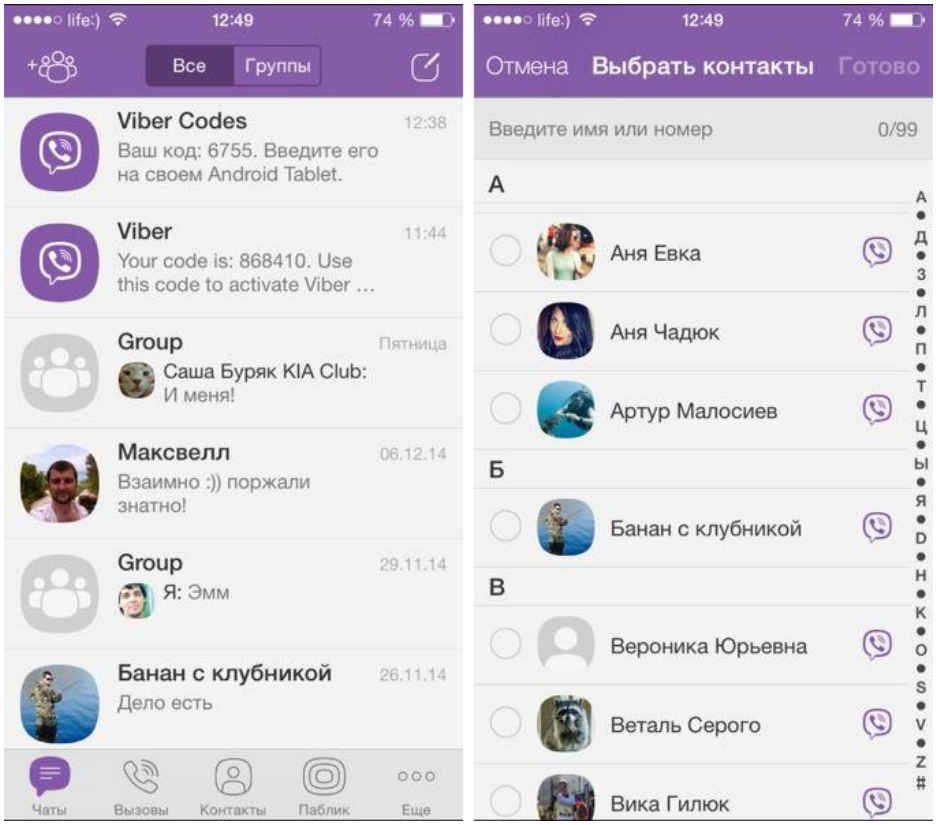

Рисунок 3 – Интерфейс мессенджера Viber

1.1.2. Корпоративные социальные сети

Краткая характеристика корпоративных социальных сетей: «Корпоративная социальная сеть, по сути, очень похожа на обычную, но с расширенными возможностями, полезными функциями для компаний.» [4].

Битрикс24, рисунок 4, платформа для совместной работы с CRM, контакт-центром, инструментами управления задачами и проектами, разработки сайтов и интернет-магазинов, автоматизации бизнес-процессов. Облачный сервис для командной работы включает чат, видеоконференции HD до 48 участников, календарь, соцсеть компании, блок новостей, Базу знаний. Соцсеть компании служит центром коммуникаций между сотрудниками в

режиме реального времени с возможностью приглашения коллег по ссылке, email, через SMS. Предусмотрено создание рабочих групп для совместной работы над проектами с настройкой прав доступа на основе ролей и гостевым режимом. Встроены схема визуальной структуры компании, списки сотрудников, учет рабочего времени, отчетность.

Преимущества:

– многоуровневая система безопасности;
– выбор между облачной и коробочной версиями;
– интеграция с 1С;
– огромный маркетплейс приложений с продуктами под разные задачи.

Недостатки:

– нельзя связаться с представителями сервиса без оплаты подписки;
– нет поддержки по телефону;
– длительное обучение;
– масштабные обновления интерфейса без возможности откатиться на старую версию.

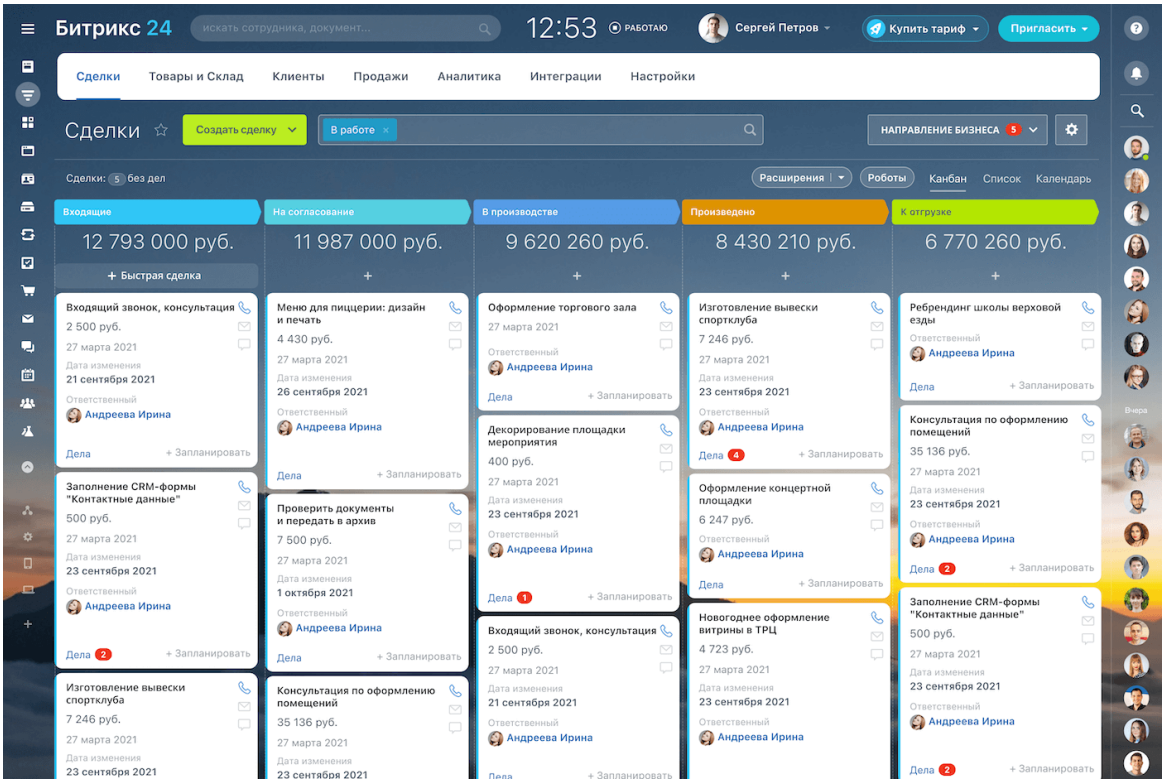

Рисунок 4 – Интерфейс корпоративной сети Битрикс24

Daoffice, рисунок 5, платформа для коммуникаций, совместной работы и управления персоналом. Разработана для создания собственной корпоративной социальной сети и брендированных мобильных приложений. Поддерживает функции совместной работы над документами, публикации постов, опросов, ленту новостей, онлайн-чаты, подключение внешних пользователей в гостевом режиме. Платформа дополнена инструментами адаптации и обучения сотрудников, включая умный поиск, конструктор страниц, тестирование, рекомендации, оргструктуру. Система вовлеченности и мотивации включает геймификацию, награды за достижения, персонализацию профиля.

Преимущества:

– инструменты стандартных соцсетей – теги, имена, возможность ставить лайки публикациям, отображение списка рекомендованных друзей;

– начисление очков сотрудникам за публикации по актуальным темам;

– доступен для Android и iOS;

– может включать неограниченное количество пользователей;

– для крупных компаний (более 250 человек) предоставляется бесплатно.

Недостатки:

– риск утечек конфиденциальной информации;

– возможны недопонимания из-за отсутствия невербальных сигналов;

– избыточное количество сообщений может перегружать сотрудников.

Рисунок 5 – Интерфейс корпоративной сети Daoffice

Пряники, рисунок 6, платформа корпоративных коммуникаций для быстрого решения HR-задач. Объединяет корпоративную социальную сеть, геймификацию, биржу идей, мобильные приложения. Корпоративная социальная сеть включает модули Живая лента, Справочник сотрудников, Карточки сотрудников, Группы. Живая лента служит коммуникационным центром для новостей, обсуждений, обмена документами, файлами. Справочник сотрудников – онлайн-каталог с контактными данными, профилями работников. Предусмотрена работа в открытых, закрытых,

невидимых группах. Система управления контентом позволяет размещать дополнительную информацию внутри корпоративной сети, включая баннеры, рекламные блоки, wiki-страницы.

Преимущества:

– обеспечивает платформу для обмена сообщениями и документами в реальном времени;

– использует элементы геймификации для повышения мотивации сотрудников;

– поддерживает различные варианты внедрения, включая облачные решения и установку на клиентский сервер;

– способствует созданию сплоченного коллектива и улучшению корпоративной культуры.

Недостатки:

– мотивация через геймификацию может отвлекать сотрудников от основных задач;

– возможно появление информационного шума и перегрузки сотрудников уведомлениями;

– сотрудники могут терять интерес к работе, фокусируясь на геймификации и рейтингах;

– требуется время на настройку системы и обучение сотрудников её использованию.

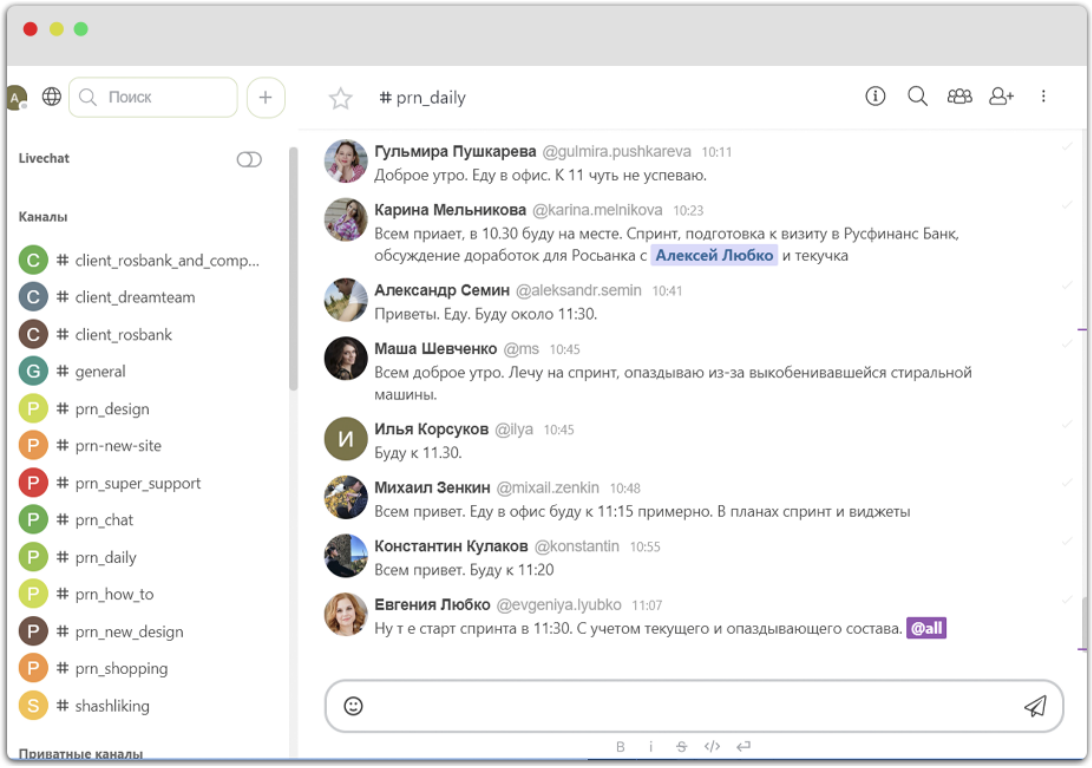

Рисунок 6 – Интерфейс корпоративной сети Пряники

1.1.3. CRM – системы

Customer Relationship Management (CRM) – это платформа, которая аккумулирует данные о клиентах, сотрудниках, процессах внутри компании и автоматизирует работу с ними. «Система управления отношениями с клиентами (CRM) представляет собой программное решение, разработанное для эффективного управления взаимодействием организации с её клиентами.» [5].

CRM-системы полезны для разных функция бизнеса и, в зависимости от их назначения, бывают трех видов. Разберём для чего нужны CRM системы и какие бывают.

CRM для продаж. Речь о единой системе хранения данных клиентов. Это упорядочивает работу менеджеров и позволяет хранить всю историю взаимодействий и саму базу данных (контакты клиентов, персональные данные, история покупок и обращений). Сотруднику придет напоминание о сроке звонка клиенту, история продаж подскажет, какой продукт или сервис лучше предложить именно сейчас. Кроме того, CRM-платформы помогают

лучше выстраивать аналитику — фиксировать воронку продаж, анализировать клиентский путь, эффективность менеджеров.

CRM для маркетинга. Маркетологам тоже важно иметь единую автоматизированную платформу для аналитики. На основе больших клиентских данных можно делать сегментацию, чтобы выстраивать более эффективную таргетированную коммуникацию. Важно анализировать рекламные кампании в разрезе по каналам. Например, сколько приносит лидов и продаж контекст, социальные сети и охватная реклама. Чтобы считать и регулярно отслеживать ключевые маркетинговые метрики (CPA, ROI, LTV и пр.), тоже нужно собирать и хранить данные на единой платформе. Прямые коммуникации с клиентами (рассылки, чат-боты) тоже завязаны на CRM-систему.

CRM для клиентского обслуживания. CRM для маркетинга работает в большей степени на привлечение новых клиентов, а CRM для продаж – на продажи этим клиентам. Здесь речь о том, чтобы улучшить процесс обслуживания и работу с текущими клиентами. Это нужно службам поддержки, административному персоналу. CRM-система автоматизирует консультации клиентов, напоминает о незавершенной заявке или доставке, собирает отзывы.

Есть несколько видов CRM-решений в зависимости от их функционала. Важно четко понимать задачи, которые будут решаться с помощью CRM-платформы, чтобы корректно выбрать один из этих типов систем.

Разберём несколько видов CRM по уровню обработки информации.

Операционная, рисунок 7. Главная задача такой системы – автоматизация рутинных задач (операций). Сбор и актуализация данных клиентов, поиск соответствующего сотрудника для обслуживания клиента, маршрутизация задач внутри команды продаж, автоматизация документооборота и отчетности. Например, система 1С:CRM позволяет выставлять счета и контролировать платежи, автоматизирует работу службы

поддержки клиентов, фиксирует активности по клиентам. Сейчас уже чисто операционных систем практически нет, поскольку есть потребность в комплексных продуктах с аналитическими и статистическими функциями.

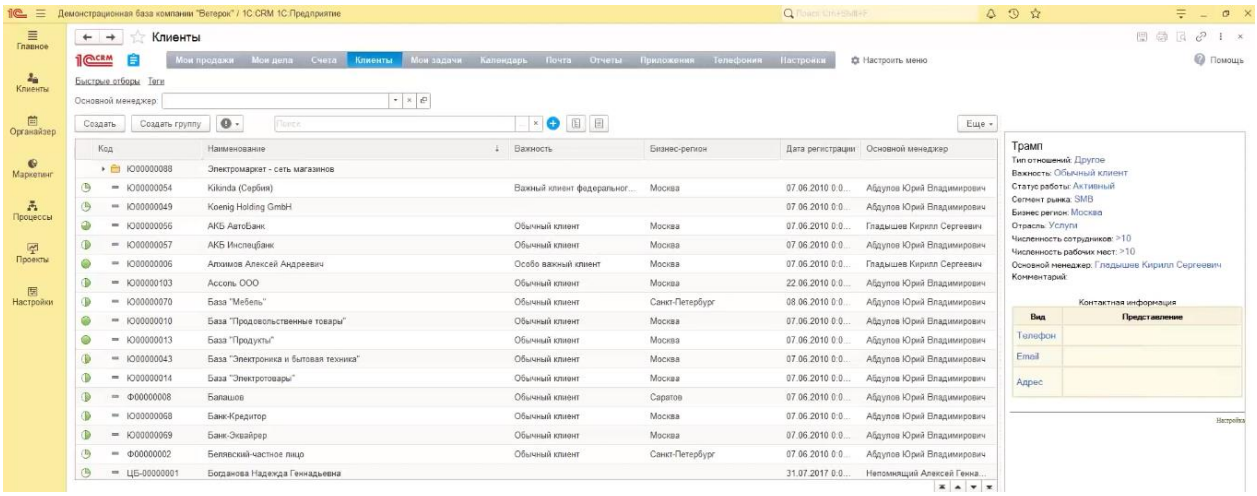

Рисунок 7 – Интерфейс 1С:CRM, раздел «Клиенты»

Аналитическая, рисунок 8. Этот вид систем используется чаще в маркетинге и аналитике, и он не связан напрямую с обслуживанием клиентов. Вы можете сегментировать клиентскую базу, анализировать каналы привлечения и весь дальнейший путь клиентов, прогнозировать бизнес-результат. Такая CRM-система нужна, когда у вас уже полностью решены оперативные вопросы, налажены процессы сбора и хранения данных, автоматизированы продажи. Следующий шаг – аналитика и улучшение бизнес-процессов, финансового и маркетингового планирования. Примером тут может быть одна из самых известных платформ – SAP CRM. Она прогнозирует и управляет воронкой продаж в режиме реального времени.

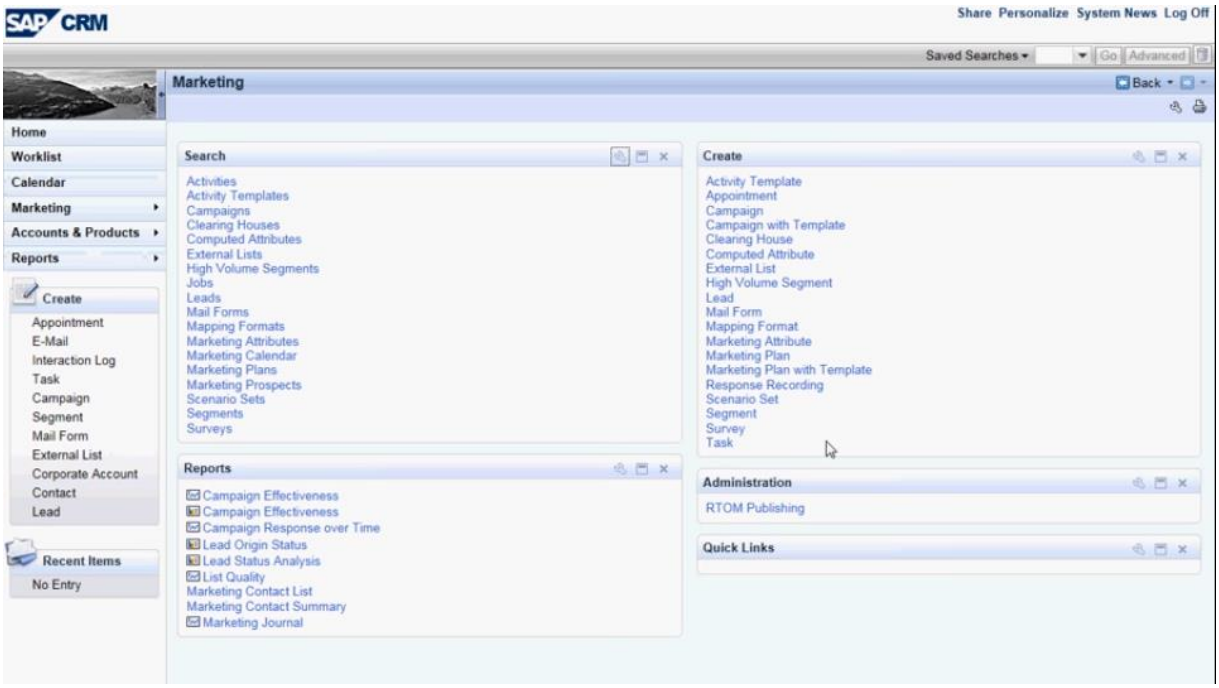

Рисунок 8 – Интерфейс SAP CRM, раздел «Маркетинг»

Коллаборационная, рисунок 9. Такая CRM-система регистрирует любую обратную связь от клиентов (запросы, отзывы, заметки). В рамках единой платформы этими данными пользуются все, кто задействован в работе с клиентами: торговые представители, клиентские менеджеры и специалисты службы поддержки. Компания оптимизирует работу отделов, а клиент получает стройную и логичную коммуникацию. Например, Sage CRM консолидирует клиентские данные и устраняет дублирование работы разных сотрудников компании.

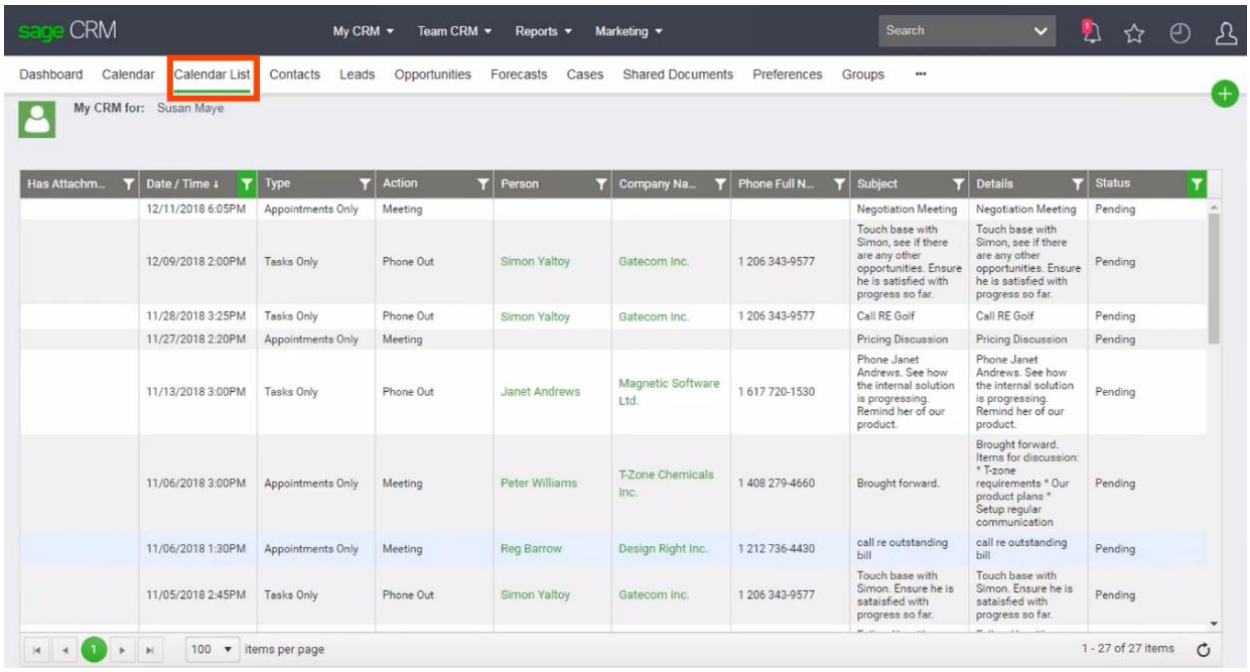

Рисунок 9 – Интерфейс Sage CRM

Преимущества:

– своевременное информирование существующих и потенциальных клиентов;

– улучшение коммуникаций внутри компании;

– сокращение времени на закрытие сделки.

Недостатки:

– расходы. Чтобы обеспечить эффективную работу, компании придется привлечь дополнительного специалиста, который займется ее обслуживанием. Кроме того, нужно позаботиться о хранении данных, что тоже может увеличить расходы;

– обучение. Если это малый бизнес, обучение для него не так актуально. При этом крупным компаниям придется проводить тренинги для работников;

– сопротивление персонала. Сотрудники могут не осознавать очевидных плюсов CRM. В результате владельцам бизнеса и менеджерам придется преодолевать сопротивление персонала.

1.2. Выбор системы управления корпоративными коммуникациями для реализации новостной ленты

Для того чтобы выбрать подходящую систему управления корпоративными коммуникациями для реализации, сопоставим преимущества и недостатки, всех рассмотренных вариантов, а также выделим самые приоритетные задачи для решения.

Задачи:

– создание и управление каналами для мгновенной рассылки новостей, обновлений и важной информации;

– координация событий, рассылка приглашений и напоминаний о мероприятиях через мессенджеры;

– низкие затраты на внедрение и эксплуатацию;

– быстрое развертывание;

– простое обучение.

Для решения таких задач нам подойдут мессенджеры, с их помощью возможно легко реализовать рассылку новостей, как вручную, так и с помощью бота, в качестве парсера новостей. Также для пользователей мы можем реализовать, чат – бота в качестве обратной связи и бота предложения новостей от пользователей. Выберем подходящий мессенджер из трёх представленных, а именно из Telegram, Viber и WhatsApp. Сравним преимущества и недостатки каждого из них.

1.2.1. Сравнение мессенджеров на основе общей информации

Разберём общую информацию о каждом мессенджере.

Все три мессенджера доступны на ПК (Персональный Компьютер), также они все распространяются бесплатно. Из всех представленных мессенджеров, только в Viber’е есть реклама, это является недостатком, так как реклама в приложениях может быть отвлекающей и навязчивой.

Приложения и блоги без рекламы является более эстетичным и приятными при использовании, что помогает контенту лучше выделяться.

Все мессенджеры совместимы с системами Android и IOS.

Viber не может быть персонализирован, в нём вы не можете выбирать различные темы или цвета.

Viber не поддерживает виджеты, а это дает больше возможностей для работы и возможность для просмотра информации, не заходя в приложение.

Из всех мессенджеров только Telegram бизнес ориентирован, это означает, что оно был разработано специально для профессионального использования. Данное приложение предлагает более серьезный дизайн, часто с учетом особенностей работы и повышенной безопасностью.

1.2.2. Сравнение безопасности мессенджеров

Сравним все мессенджеры на основе безопасности.

Все мессенджеры имеют двустороннее шифрование, которое защищает данные между двумя взаимодействующими сторонами и гарантирует, что только соответствующий получатель может получить доступ к незашифрованной информации.

Сообщения по умолчанию зашифрованы в Viber и WhatsApp. По умолчанию для всех чатов включено сквозное шифрование, что отличается от других приложений, которые шифруют сообщения только в том случае, если пользователь специально включает данную функцию.

Функцию обнаружения скриншотов имеет только Telegram. Обнаружение скриншотов – это функция безопасности, которая позволяет пользователям получать уведомления, когда кто-то делает снимки их сообщений. Функция обнаружения скриншотов экрана, часто используемая в приложениях для безопасного обмена сообщениями, обеспечивает безопасность ваших сообщений и данных.

Telegram и WhatsApp имеют сквозное шифрование в групповых чатах.

Настройки конфиденциальности имеют все мессенджеры.

В мессенджере Viber не нужно иметь сим-карту, чтобы войти. Приложение не подключено на ваш номер телефона, так что вы можете использовать его на устройствах, таких как планшетники, не нуждаясь в SIM-карте.

Любое из этих приложений требует от пользователя чтобы создать аккаунт.

Telegram и WhatsApp имеют возможность установить пароль, чтобы сохранить вашу личную информацию.

Telegram можно использовать анонимно, т.е. вам не надо использовать своё настоящее имя.

1.2.3. Сравнение функции обмена сообщения мессенджеров

Во всех мессенджерах вы можете увидеть, когда кто – то открыл ваше сообщение, что кто – то печатает, посмотреть, когда пользователь был активным, посылать звуковые голосовые файлы, отправлять временные текстовые сообщения, есть возможность удалять историю сообщений, оставаться невидимым, функция «Поделиться местоположением».

В мессенджерах Telegram и Viber вы можете указать информацию о доступности. Например, вы заняты или находитесь далеко. Это идеальный вариант для того случая, когда вы работаете, но хотите остаться доступными для важных сообщений.

Все мессенджеры имеют двусторонний видео чат, групповой видео чат, двусторонний голосовой чат и имеют групповой голосовой чат.

Ни один из мессенджеров не имеет интеграцию входящих смс. Сообщения от приложения могут храниться в ваших обычных входящих СМС, поэтому вы можете видеть всю вашу переписку в одном месте.

Только Telegram имеет функцию отправки сообщений по графику, вы можете написать сообщение и запланировать его отправку в определенное время, например: послать кому-нибудь текст, чтобы разбудить его с утра.

1.2.4. Сравнение совместного использования контента в мессенджерах

Во всех мессенджерах вы можете отправлять документы, видео, музыкальные файлы, рисовать на файлах, добавлять подписи и заголовки к вашим фотографиям, экспортировать ваши данные на электронную почту, отправлять графические сообщения и делать фотографии внутри приложения.

Только Telegram и Viber имеют возможность предварительного просмотра изображения и имеют встроенный переводчик.

Ни один из мессенджеров не могут архивировать старые данные и не имеют голосовых фильтров.

Самый большой размер файла, которым вы можете поделиться в мессенджере Telegram 2000MB, в Viber 200MB, а в WhatsApp всего 100MB. Чем выше предел, тем меньше вы ограничены в обмене контентом.

1.2.5. Сравнение контактов мессенджеров

Все мессенджеры имеют интеграцию существующих контактов, могут посылать контактную информацию целиком.

Мессенджер WhatsApp не имеет возможности связываться с любым пользователем приложения.

Только Viber имеет настройку с избранными контактами и только Telegram имеет функцию «Двусторонний отказ», это означает, что вы должны добавить или принять контакт, прежде чем он сможет посылать вам сообщения. Это предотвращает вас от получения нежелательных сообщений от незнакомцев.

1.2.6. Предварительные результаты сравнения

На основе сравнения ключевых функций, Telegram является для нас лучшим выбором, потому что приложение бизнес ориентировано, использует функцию «Двусторонний отказ», имеет функцию отправки сообщений по графику, максимальный размер файла 2000 MB.

1.2.7. Сравнение охвата аудитории и распространенность мессенджеров

В данном сравнении мы будем использовать открытые данные из источника DataReportal, этот источник предлагает тысячи бесплатных отчетов, которые помогут понять, что люди делают в интернете.

Соберём все последние данные [13-16] как люди в Российской Федерации используют цифровые устройства и сервисы за январь 2021, 2022, 2023 и 2024 года.

2021 год:

– население страны составило 145,9 млн человек;

– насчитывалось 124,0 млн пользователей интернета;

– в период с 2020 по 2021 год количество интернет – пользователей увеличилось на 6,0 млн (+5,1%);

– распространение интернета составило 85,0%;

– насчитывалось 99,00 млн пользователей социальных сетей.

2022 год:

– население страны составило 145,9 млн человек;

– насчитывалось 129,8 млн интернет – пользователей;

– число интернет – пользователей увеличилось на 5,8 млн (+4,7%);

– распространение интернета составило 89,0%;

– насчитывалось 106,0 млн пользователей социальных сетей.

2023 год:

– население страны составило 145,7 млн человек;

– насчитывалось 127,6 млн интернет – пользователей;

– число интернет – пользователей увеличилось на 5,8 млн (+4,7%);

– распространение интернета составило 88,2%;

– насчитывалось 106,0 млн пользователей социальных сетей.

2024 год:

– население страны составило 144,2 млн человек;

– насчитывалось 130,4 млн интернет – пользователей,

– распространение интернета составляло 90,4%;

– насчитывалось 106,0 миллиона пользователей социальных сетей.

Разберём наиболее часто используемые мессенджеры. Процент интернет – пользователей в возрасте от 16 до 64 лет, которые ежемесячно пользуются каждой платформой.

Согласно статистике, собранной в январе 2021, 2022, 2023 и 2024 годов, аудитория мессенджера Viber падает. В 2021 году она составляла 42,5%, в 2022 году 40,7%, в 2023 году уже 34,7%, а в январе 2024 года она упала до отметки в 30,1%.

Аудитория мессенджера WhatsApp, каждый год изменяется на +/- 5% - 10%, так в 2021 году она составляла 75,8%, в 2022 году 80,9%, в 2023 году она упала до уровня 71,5%, а в 2024 году поднялась до уровня в 74,5%.

Из всех мессенджеров очень выделяется Telegram, рост популярности мессенджера большой, так например в 2021 году его аудитория составляла 27,4%, в 2022 году произошёл рост примерно в два раза, до 50,8% и в 2023 и 2024 году наблюдался стабильный рост в +/- 10 - 15%, 64,4% и 72,7% соответственно. Графическое представление данной статистики на рисунке 10.

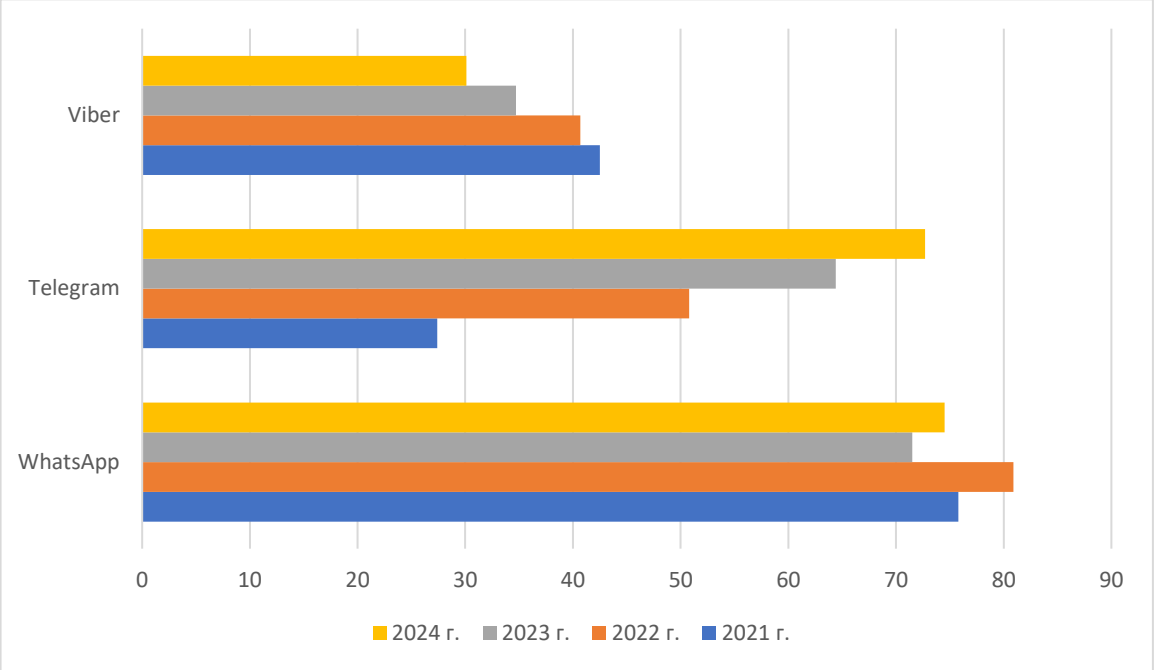

Рисунок 10 – Диаграмма наиболее часто используемые мессенджеры

Разберём любимые платформы социальных сетей, мессенджеры так же относятся к социальным сетям. Процент активных пользователей социальных сетей в возрасте от 16 до 64 лет, которые считают, что каждый из вариантов является их «Любимой» платформой для социальных сетей.

В этот раз нам придётся сократить временные рамки, до 2022 по 2024 год, так как до 2022 года источник не вёл такую статистику.

В 2022 году Viber могли назвать «Любимой» платформой 3,6%, в 2023 году уже 3,1%, а в 2024 году всего 2,8% пользователей.

WhatsApp в 2022 году могли назвать «Любимой» платформой 20,6%, в 2023 году падение было 15,4%, а в 2024 году число таких пользователей выросло до 16,2%.

По сравнению со всеми Telegram выделяется в самую лучшую сторону, в 2022 году мессенджер могли назвать «Любимой» платформой только 8,2%, в 2023 году этот результат увеличился в примерно в два раза, до 20,5%, а в 2024 году составляет 27,5%. Графическое представление данной статистики на рисунке 11.

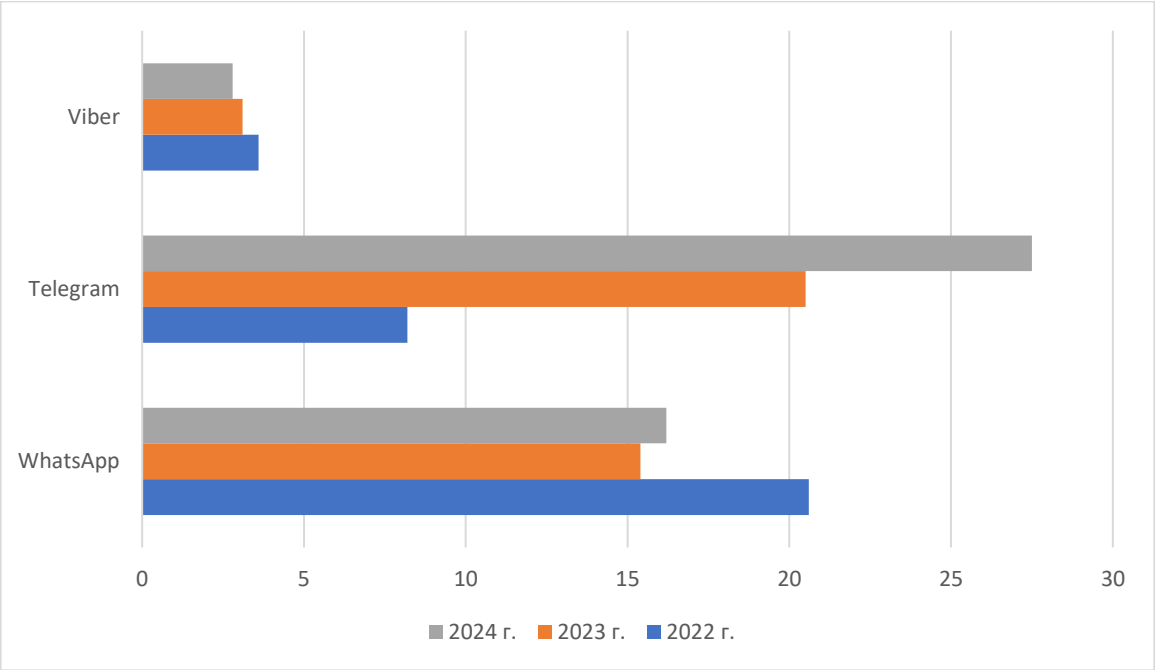

Рисунок 11 – Диаграмма любимые платформы социальных сетей

Разберём сколько пользователи проводили времени, с помощью приложений для социальных сетей. Среднее время в месяц, которое активные пользователи тратили на использование Android – приложений для каждой платформы в период с 1 июля по 30 сентября 2023 года.

На Telegram пользователи тратили 9 часов 22 минуты в месяц, а на WhatsApp всего 8 часов 15 минут.

Проанализируем ежемесячные сеансы работы с приложениями для социальных сетей. Среднее количество раз, когда активные пользователи Android – приложений для каждой платформы открывают соответствующее приложение в месяц.

WhatsApp пользователи открывают в среднем 465,7 раз, а Telegram 359,1 раз.

Для мессенджера Viber такая статистика отсутствует, так как он не попал в выборку из-за слишком низкого показателя, по сравнению с другими приложениями.

Сравним динамику текстовых запросов, по ключевым словам, Telegram и WhatsApp в поисковой системе Яндекс с на основе данных Яндекс Вордстат

[17] за январь 2021 года по апрель 2024 года. На рисунке 12 представим динамику.

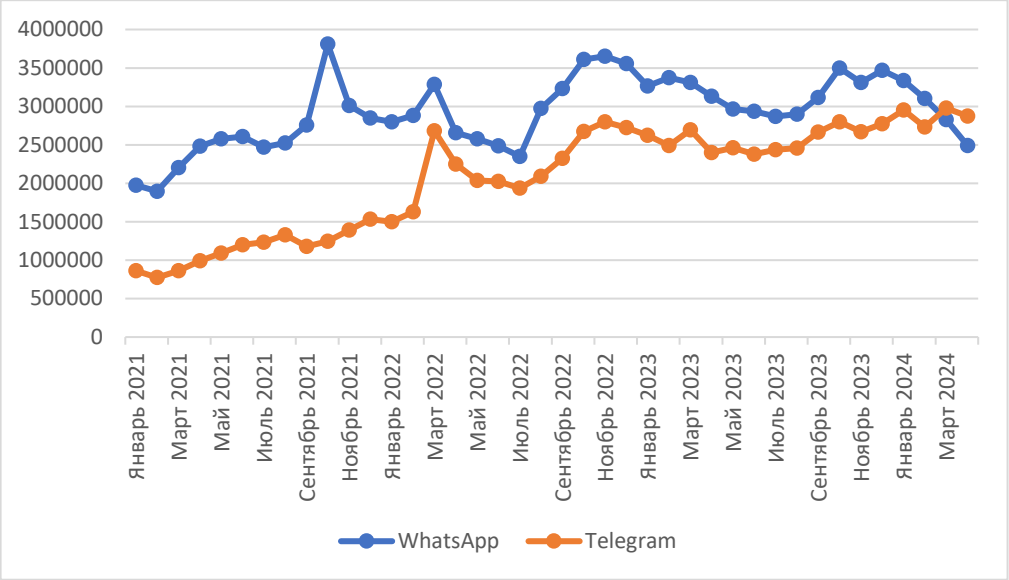

Рисунок 12 – Диаграмма динамики запросов

WhatsApp, динамика составляет от минимум 1 896 598 запросов в феврале 2021 года, до максимум 3 808 468 запросов в октябре 2021 года.

Telegram, динамика составляет от минимум 773 846 запросов в феврале 2021 года, до максимум 2 976 871 в марте 2024 года.

1.3. Обзор аналогов новостных лент в системах управления корпоративными коммуникациями

1.3.1. Мессенджеры и чаты

Во всех мессенджерах и чатах, новостные ленты реализуются с помощью ботов. Для этого используются различные языки программирования, например для Telegram используются:

– Python популярный язык программирования, «Python поддерживает различные стили программирования, включая процедурное, объектно-ориентированное и функциональное программирование.» [6]. Фреймворк python-telegram-bot;

– Node.js, «Это среда выполнения на основе JavaScript, которая позволяет разрабатывать серверные приложения с использованием языка программирования JavaScript. ». [7]. Фреймворк node-telegram-bot-api;

– Java, «Это язык программирования общего назначения, который следует парадигме объектно-ориентированного программирования и подходу «Написать один раз и использовать везде». » [8]. Библиотека TelegramBots API;

– PHP, «Это распространенный язык программирования общего назначения соткрытым исходным кодом. PHP специально сконструирован для веб-разработок и его код может внедряться непосредственно в HTML.» [9]. Библиотека php-telegram-bot;

– Go, «Простой в освоении, высоко масштабируемый язык системного программирования. Хотя Go немного анемичен, он стал популярным языком программирования за короткий период.» [10]. Библиотека telebot.

Для WhatsApp используются:

– Python: библиотеки, yowsup для неофициальных API или twilio-python для работы с Twilio API;

– Node.js: библиотека whatsapp-web.js;

– Java: официальные SDK и различные библиотеки;

– PHP: с использованием Twilio API или других сторонних сервисов.

Для Viber используются:

– Python: библиотека viber-bot-python;

– Node.js: библиотека viber-bot;

– PHP: официальные библиотеки и SDK;

– Java: использование Viber API для создания ботов.

1.3.2. Корпоративные социальные сети

Битрикс24. В ленте можно писать сообщения всем сотрудникам, группе людей или персонально, рисунок 13. Текст сообщения можно оформить в визуальном редакторе, а к сообщению прикрепить необходимый файл.

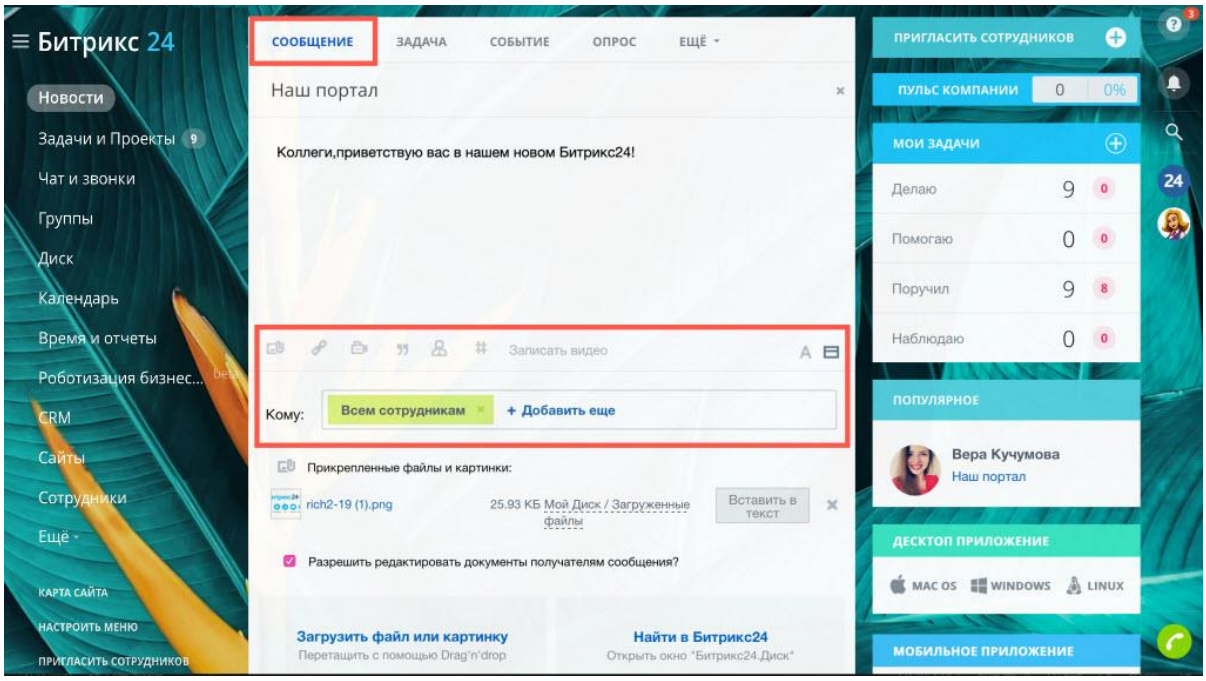

Рисунок 13 – Лента Битрикс24

Когда сообщение публикуется, вы можете видеть, сколько человек просмотрели сообщение и кто именно это сделал, рисунок 14. К сообщению можно писать комментарии и выражать свои эмоции через лайки и эмодзи.

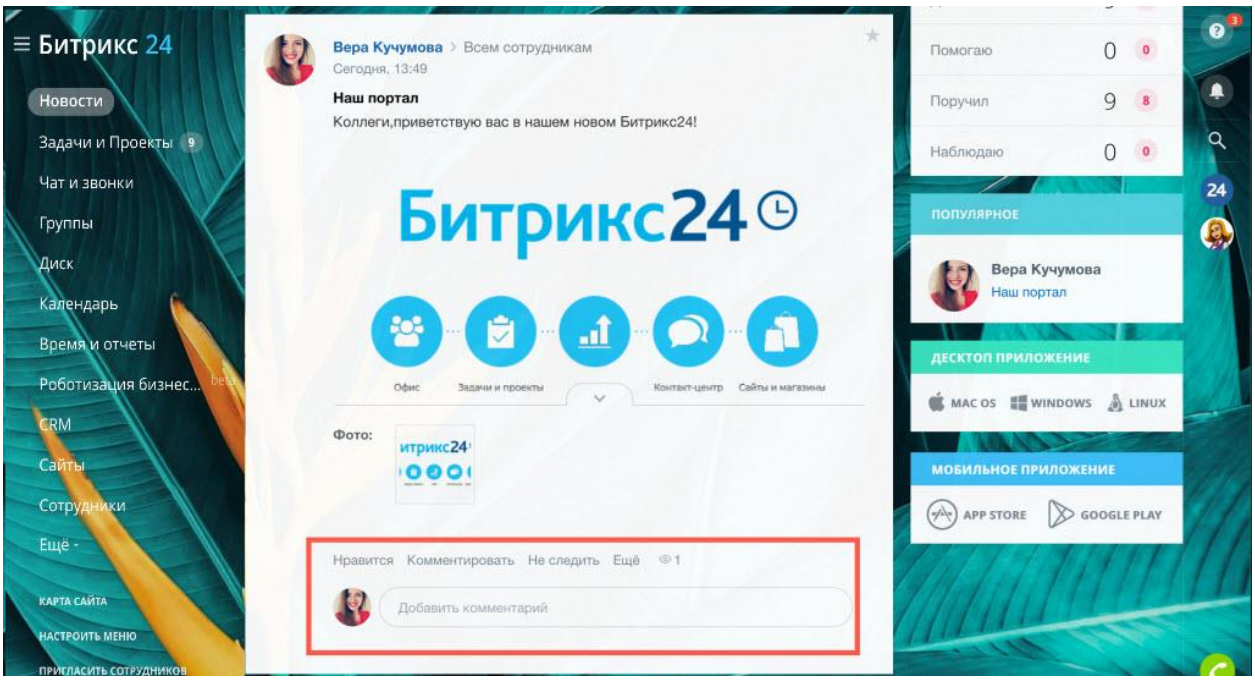

Рисунок 14 – Опубликованное сообщение

Сообщение можно сделать важным, чтобы сотрудники обратили на него особое внимание. Также через сообщение можно выразить благодарность коллегам за проделанную работу или поздравить с особым событием, например, с днем рождения.

Daoffice, рисунок 15, форматы сторис, баннеры, посты, реакции — это инструменты для формирования вовлеченного комьюнити. Традиционные новости о жизни компании становятся живым контентом, к которому можно привлечь дополнительное внимание пользователей.

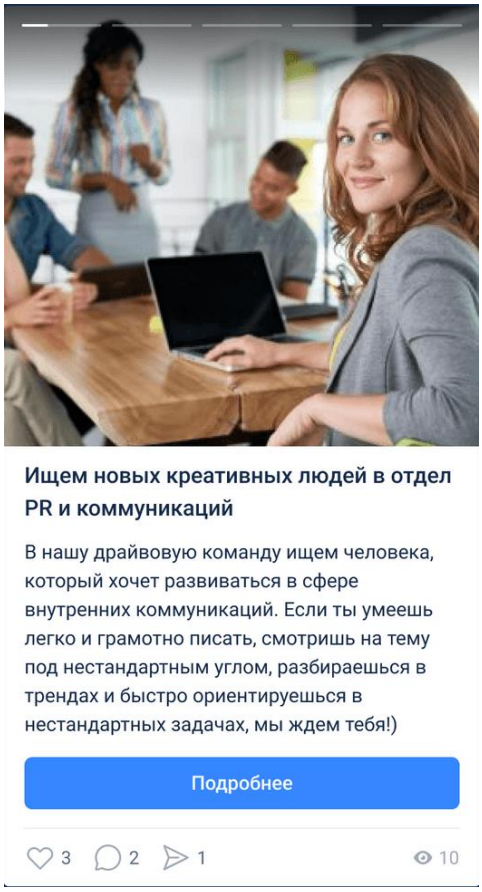

Рисунок 15 – Публикация баннера

1.3.3. CRM – системы

CRM Простой бизнес. Лента новостей, рисунок 16, открывается по нажатию на кнопку «Оповещения» в правом верхнем углу главной страницы.

В ленте новостей предоставляются следующие возможности:

– создать дело;

– комментировать;

– цитировать;

– копировать;

– копировать ссылку;

– переместить;

– изменить категорию.

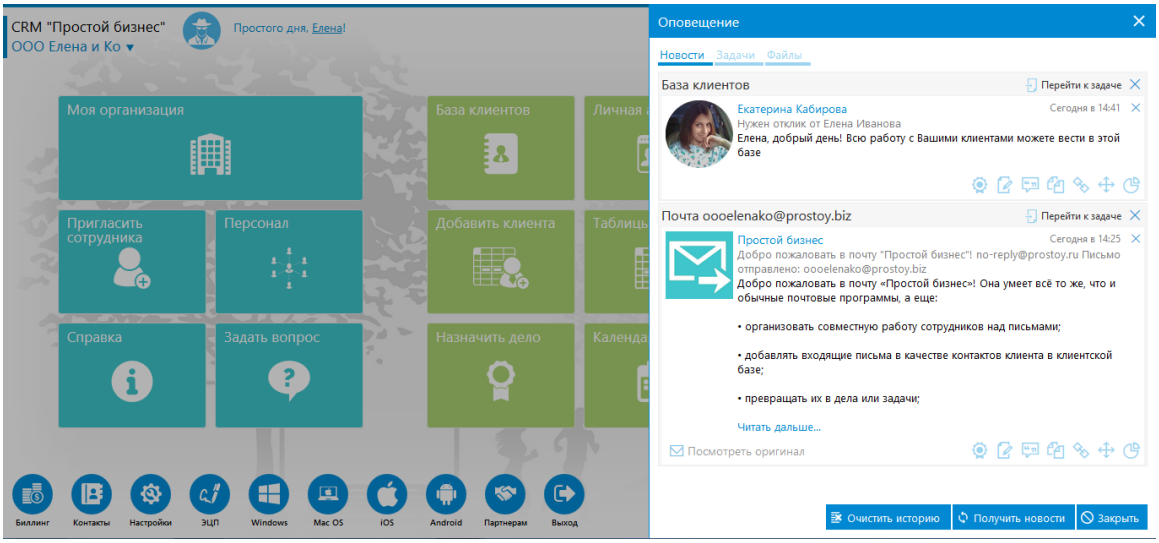

Рисунок 16 – Новостная лента CRM простой бизнес

При получении новых новостей в верхней части окна «Оповещения» появляется мигающая надпись, при нажатии на которую свежие новости отобразятся в ленте.

Внизу окна «Оповещения» расположены кнопки:

– очистить историю – скрыть все новости из ленты;

– получить новости – проверить новые оповещения;

– закрыть – закрыть окно «Оповещения».

Если новое оповещение пришло, когда на экране была открыта любая страница, кроме главной, то оповещение появится в верхнем правом углу экрана. Оповещение исчезнет с экрана само через несколько секунд, а также, если нажать по нему левой кнопкой мыши, рисунок 17.

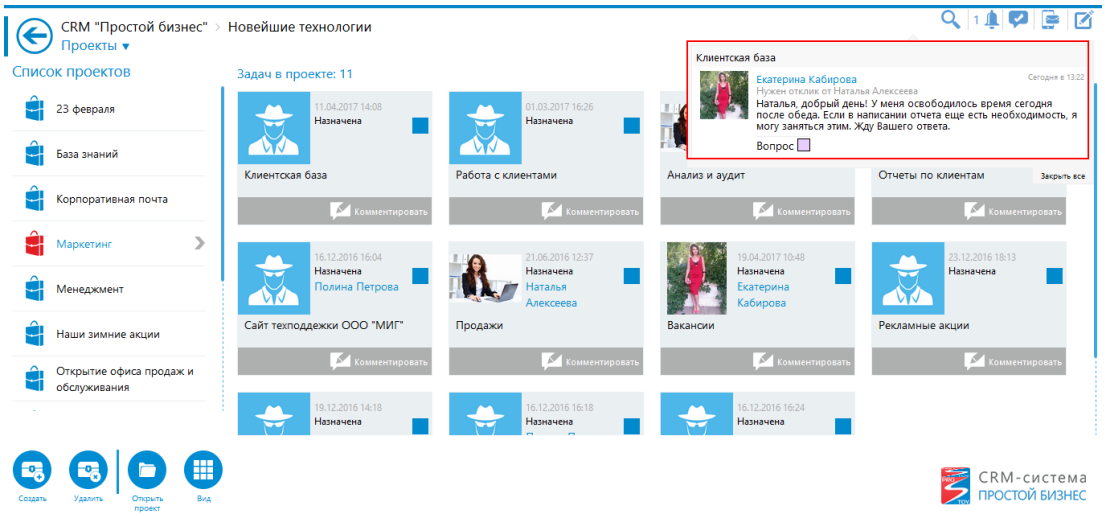

Рисунок 17 – Оповещение в CRM простой бизнес

1.4. Характеристика АО «УК ЭФКО»

Группа компаний «ЭФКО» — крупнейший производитель продуктов питания в России, системообразующее предприятие Российской Федерации, логотип на рисунке 18.

Рисунок 18 – Логотип компании

Компания специализируется на переработке масличных культур (подсолнечник, соевые бобы, рапс), производстве фасованных масел, жиров и маргаринов, майонезов, кетчупов и молочных йогуртов.

Ключевые торговые марки — «Слобода», Altero и Hi! (от англ. healthy innovation — «здоровое будущее»). Мощности производственных площадок, расположенных в Белгородской, Воронежской, Липецкой, Свердловской областях, Краснодарском крае, а также в Республике Казахстан, способны обеспечивать переработку свыше 3,5 млн т масличных в год. Активно диверсифицирует бизнес за счет развития собственного научного инновационного центра, разработки растительного мяса и молока под брендом Hi!, венчурных инвестиций в фудтех и биотехнологии.

Кроме того, «ЭФКО» реализует проект «Умная Ферма», в который входят «Дивизион эффективных кормов» и «Лаборатория эффективных кормов» — подразделения, специализирующиеся на кормовых добавках и комплексном подходе к составлению рационов и содержанию крупного рогатого скота.

В Алексеевской слободе, где в конце 20-х годов XIX века впервые на территории Российской империи было получено масло из семечек подсолнечника, и был построен первый маслобойный завод в России.

Спустя почти 200 лет Алексеевка остается крупнейшим центром переработки подсолнечника в стране. Именно здесь, в Алексеевке, находится крупнейшая производственный комплекс «ЭФКО» и штаб-квартира Совета директоров компании. Офис компании в Воронеже, рисунок 19.

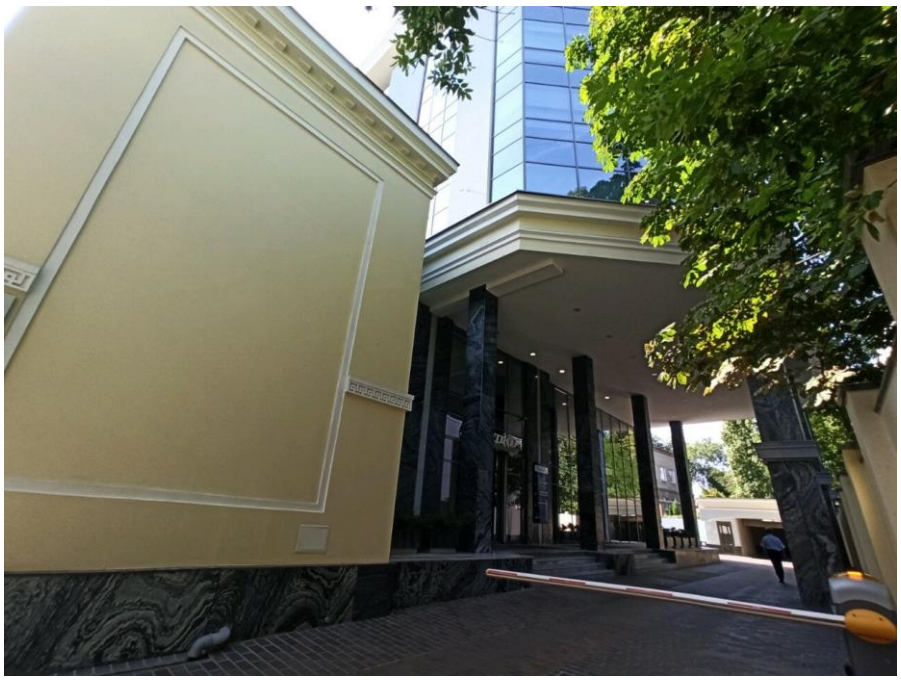

Рисунок 19 – Офис компании в Воронеже

1.5. Структура «ЭФКО»

На вершине иерархии находится директор, который отвечает за общее руководство организацией, рисунок 20. Под ним находятся несколько отделов, каждый из которых возглавляет свой руководитель.

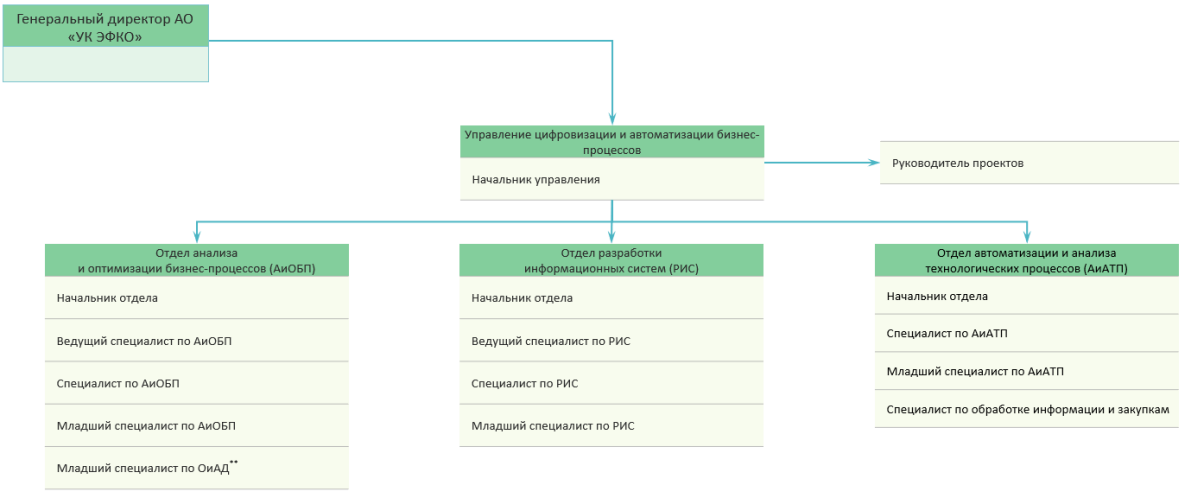

Рисунок 20 – Структурная организация «ЭФКО»

Основные отделы организации:

Управление цифровизацией и автоматизацией бизнес-процессов:

– начальник управления, руководит управлением и отвечает за стратегическое планирование и внедрение цифровизации и автоматизации в компании;

– руководитель проектов, координирует и управляет проектами, связанными с цифровизацией и автоматизацией, обеспечивает их успешное выполнение в установленные сроки.

Отдел анализа и оптимизации бизнес-процессов (АиОБП):

– начальник отдела, руководит отделом и отвечает за общую стратегию и выполнение задач отдела;

– ведущий специалист по АиОБП, осуществляет анализ и оптимизацию существующих бизнес-процессов, разрабатывает предложения по их улучшению;

– специалист по АиОБП, проводит исследования и анализ данных, участвует в разработке и внедрении решений для оптимизации бизнес-процессов;

– младший специалист по АиОБП, поддерживает старших специалистов, выполняет анализ данных и готовит отчеты;

– младший специалист по ОиДА, Занимается общими задачами отдела анализа и оптимизации данных, поддерживает процессы обработки информации и их улучшения.

Отдел разработки информационных систем (РИС):

– начальник отдела, руководит отделом и отвечает за разработку и внедрение информационных систем в компании;

– ведущий специалист по РИС, участвует в проектировании и разработке информационных систем, координирует работу команды разработчиков;

– специалист по РИС, проводит разработку и тестирование информационных систем, обеспечивает их интеграцию и поддержку;

– младший специалист по РИС, поддерживает разработчиков в тестировании и внедрении информационных систем, занимается решением технических задач.

Отдел автоматизации и анализа технологических процессов (АиАТП):

– начальник отдела, руководит отделом и отвечает за автоматизацию и анализ технологических процессов;

– специалист по АиАТП, проводит автоматизацию технологических процессов, анализирует их эффективность и предлагает улучшения;

– младший специалист по АиАТП, поддерживает старших специалистов, участвует в автоматизации и анализе технологических процессов;

– специалист по обработке информации и закупкам, отвечает за обработку информации, связанную с закупками, и оптимизацию процессов закупок.

1.6. Постановка задачи

1.6.1. Цель и назначение автоматизированного варианта решения задачи

Цель автоматизации состоится в улучшении системы внутренней коммуникации и оповещения сотрудников компании. Подцели включают.

Косвенный эффект в управлении организацией:

– повышение эффективности внутренней коммуникации;

– увеличение числа информированных сотрудников;

– уменьшение временных затрат на распространение информации.

Прямой эффект, отражающийся на себестоимости выпускаемой продукции или оказываемых услуг:

– сокращение времени и стоимости обработки информации;

– повышение степени достоверности обработки информации;

– повышение степени автоматизации получения первичной информации;

– увеличение количества аналитических показателей, получаемых на базе исходных данных.

1.6.2. Назначение решения задачи

Решение задачи включает автоматизацию следующих функций управления и операций обработки данных:

Автоматизация процессов:

– сбор новостей из различных источников и их автоматическая публикация;

– управление обращениями сотрудников (тикетами);

– прием и модерация предложений новостей от сотрудников.

Эффективное управление информацией:

– разделение функций ботов для управления различными видами информации;

– обеспечение оперативного обновления данных и уведомлений.

Удобство для пользователей:

– интерактивное взаимодействие через привычный интерфейс Telegram;

– легкость подачи обратной связи и предложений.

Безопасность и контроль:

– использование встроенных механизмов Telegram для защиты данных;

– возможность контроля и управления действиями ботов администраторами.

Гибкость и масштабируемость:

– легкость адаптации ботов под изменяющиеся потребности компании;

– возможность добавления новых функций и интеграций без существенных изменений в инфраструктуре.

1.6.3. Общая характеристика организации решения задачи на ЭВМ

Архитектура аппаратной платформы.

Предполагаемая архитектура включает использование модели клиент-сервер, где:

– серверная часть отвечает за выполнение логики работы ботов и хранение данных;

– клиентская часть представлена интерфейсом Telegram для взаимодействия пользователей с ботами.

Изменения в функциях подразделения:

– внедрение ботов изменит функции подразделения, связанного с внутренней коммуникацией, сбором и обработкой информации;

– автоматизация рутинных задач позволит сотрудникам сосредоточиться на более сложных и творческих задачах.

Источники поступления информации:

– оперативная информация поступает через RSS-каналы для новостного бота и от сотрудников через ботов для предложений новостей и тикетов, «RSS — это широко используемая технология передачи информации через Интернет и интрасети. Многие веб-сайты предлагают RSS-каналы, на которые можно подписаться для автоматического получения последних сведений.» [12];

– условно – постоянная информация включает настройки ботов и данные модераторов, которые обновляются по мере необходимости.

Этапы решения задачи.

Проектирование системы:

– анализ требований и определение функционала ботов;

– разработка архитектуры системы.

Разработка и тестирование:

– написание кода ботов;

– тестирование на корректность работы и безопасность.

Внедрение и обучение:

– внедрение системы в рабочую среду;

– обучение сотрудников использованию ботов.

Эксплуатация и поддержка:

– регулярное обновление системы;

– поддержка пользователей и решение возникающих проблем.

Порядок ввода первичной информации:

– для efkoparsnews_bot: ввод RSS-каналов и настройка параметров публикации;

– для efkochat_bot: регистрация пользователей и настройка модераторов;

– для NewsFeedEfko_bot: настройка каналов для предложений новостей и модерации.

Результаты и их использование:

– автоматически обновляемые новостные ленты для сотрудников;

– управление обращениями и тикетами, что позволяет быстро решать проблемы;

– модерация и управление предложениями новостей для поддержания актуальности информации.

Система ведения файлов в базе данных:

– базы данных для хранения новостей, тикетов и предложений;

– регулярное обновление данных и поддержка их целостности и безопасности.

Режим решения задачи:

– диалоговый режим работы с использованием методов телеобработки для взаимодействия пользователей с ботами;

– постоянная работа ботов в реальном времени.

Периодичность решения задачи:

– постоянная работа ботов для оперативного обновления и обработки информации в режиме реального времени.

1.7. Экономическая сущность задачи

Для расчета экономической сущности задачи автоматизации процессов с помощью ботов необходимо определить следующие показатели:

– сокращение затрат на обработку информации. Время, затрачиваемое на обработку информации до и после автоматизации. Стоимость рабочего времени сотрудников;

– повышение эффективности взаимодействия. Количество обработанных тикетов до и после автоматизации. Увеличение производительности труда;

– снижение операционных издержек. Затраты на разработку и внедрение ботов. Экономия на заработной плате сотрудников за счет автоматизации рутинных задач.

1.7.1. Исходные данные

– часовая ставка сотрудников: 500 рублей в час;

– время, затрачиваемое на обработку одного тикета вручную: 30 минут;

– количество тикетов в день до автоматизации: 20 тикетов;

– время на разработку и внедрение ботов: 200 часов;

– затраты на оборудование и ПО: 50 000 рублей;

– ожидаемое сокращение времени обработки тикета после автоматизации: до 5 минут.

1.7.2. Формулы для расчета

Затраты, руб/мес, на обработку тикетов до автоматизации:

$$\text{Затраты}_{\text{до}} = \text{A} \times \text{B} \times \text{C} \times \text{D}, \quad (1)$$

Затраты, руб/мес, на обработку тикетов после автоматизации:

$$\text{Затраты}_{\text{после}} = \text{A} \times E \times \text{C} \times \text{D}, \quad (2)$$

A – количество тикетов в день;

B – время на обработку одного тикета, час;

C – часовая ставка сотрудников, руб/час;

D – 22 рабочих дня в месяц;

E – время на обработку одного тикета после автоматизации.

Экономия на заработной плате сотрудников:

$$\text{Экономия} = \text{Затраты}_{\text{до}} - \text{Затраты}_{\text{после}}, \quad (3)$$

Общие затраты на автоматизацию:

$$\text{Общие затраты} = \text{F} \times \text{G} + \text{H}, \quad (4)$$

F – время на разработку и внедрение ботов, часов;

G – часовая ставка сотрудников, руб/час;

H – затраты на оборудование и ПО.

Возврат инвестиций (ROI):

$$ROI = \left(\frac{\text{Экономия} - \text{Общие затраты}}{\text{Общие затраты}}\right) \times 100\%, \quad (5)$$

1.7.3. Расчеты

Затраты на обработку тикетов до автоматизации расчет по формуле 1:

$$20 \times 0.5 \times 500 \times 22 = 110000 - \text{руб.}$$

Затраты на обработку тикетов после автоматизации расчет по формуле 2:

$$20 \times \frac{5}{60} \times 500 \times 22 = 18333 - \text{руб.}$$

Экономия на заработной плате сотрудников расчет по формуле 3:

$$110000 - 18333 = 91667 - \text{руб./мес}$$

Общие затраты на автоматизацию расчет по формуле 4:

$$200 \times 500 + 50000 = 150000 - \text{руб.}$$

Возврат инвестиций (ROI) расчет по формуле 5:

$$\left(\frac{91667 - 150000}{150000}\right) \times 100\% = -38.89\%$$

На основании проведенных расчетов видно, что:

– сокращение затрат на обработку информации составляет 91 667 рублей в месяц;

– общие затраты на автоматизацию составляют 150 000 рублей;

– возврат инвестиций (ROI) составляет -38.89% на первый месяц.

Экономия на заработной плате сотрудников начинает покрывать затраты на автоматизацию после примерно 1.64 месяцев работы (150 000 / 91 667). Таким образом, через 2 месяца проект начнет приносить чистую экономию. Эти показатели подчеркивают экономическую целесообразность автоматизации процессов с помощью Telegram-ботов в долгосрочной перспективе.

1.8. Формализация расчетов подзадач

1.8.1. Общая структура задачи

Автоматизация процессов с использованием ботов включает в себя подзадачи:

– парсинг новостей и их автоматическая публикация;

– управление тикетами (обращениями пользователей);

– прием предложений новостей от пользователей и модерация.

1.8.2. Парсинг новостей и автоматическая публикация (efkoparsnews_bot)

Входные данные:

– список RSS-каналов.

Выходные данные:

– отформатированные новости, готовые для публикации.

Основные шаги:

1) инициализация:

– загрузка списка RSS-каналов и настроек из переменных окружения;

– инициализация базы данных SQLite для хранения данных о модераторах и опубликованных ссылках.

2) парсинг новостей:

– использование библиотеки feedparser для извлечения новостей.

3) фильтрация и форматирование:

– проверка, была ли ссылка уже опубликована;

– форматирование времени публикации в московское время.

4) публикация новостей:

– отправка новостей в канал Telegram, если они еще не были опубликованы.

1.8.3. Управление тикетами, обращениями пользователей (efkochat_bot)

Входные данные:

– сообщения от пользователей.

Выходные данные:

– созданные тикеты и обновления статусов.

Основные шаги:

1) инициализация:

– настройка бота и базы данных для хранения тикетов.

2) создание тикета:

– получение сообщения от пользователя и сохранение его в базу данных.

3) обновление тикета:

– изменение статуса тикета при его обработке модератором.

4) уведомление пользователей:

– отправка уведомлений пользователям о статусе их тикетов.

1.8.4. Прием предложений новостей от пользователей и модерация (NewsFeedEfko_bot)

Входные данные:

– предложения новостей от пользователей (текст, документы, фото и видео).

Выходные данные:

– одобренные или отклоненные новости.

Основные шаги:

1) инициализация:

– настройка бота для приема сообщений и файлов от пользователей.

2) прием предложений:

– получение предложений новостей от пользователей.

3) модерация новостей:

– отправка предложенных новостей на модерацию.

4) уведомление пользователей:

– уведомление пользователей о статусе их предложений (одобрение или отклонение с указанием причины).

Эти шаги допускают формализовать процессы автоматизации, управления тикетами и модерации новостей, обеспечивая эффективное выполнение задач, таблица 1 «Формализованное описание входных показателей» и таблица 2 «Формализованное описание результатных показателей».

Таблица 1 – Формализованное описание входных показателей

Наименование входного показателя	Идентификатор входного показателя	Описание
Список RSS-каналов	rss_channels	URL-адреса каналов для парсинга новостей
Сообщения от пользователей	user_messages	Тексты сообщений, отправленные пользователями
Предложения новостей	news_proposals	Тексты, документы, фото и видео, отправленные пользователями
Данные тикетов	ticket_data	Информация о тикетах, созданных пользователями

Таблица 2 – Формализованное описание результатных показателей

Наименование результатного показателя	Идентификатор результатного показателя	Алгоритм расчета
Отформатированные новости	formatted_news	Парсинг и фильтрация новостей
Созданные тикеты	created_tickets	Запись в базу данных тикетов
Обновленные статусы тикетов	updated_ticket_status	Обновление статуса тикета
Уведомления пользователям и модераторам	notifications	Отправка сообщений через Telegram API

2. ПРОЕКТНЫЙ РАЗДЕЛ

2.1. Информационное обеспечение задачи

2.1.1. Информационная модель и ее описание

Основные сущности и их описание.

Модератор новостей:

1) новость, состоит из содержания, даты публикации и автора;

2) статус новости, может быть, как ожидает проверки, одобрена или отклонена;

3) пользователи, список авторов и модераторов.

Бот – парсер новостей:

1) источники новостей, RSS – каналы;

2) новости, состоят из заголовка, ссылки на сайт источник, краткое описание и дата публикации;

3) история публикаций, сохраненные ссылки на предыдущие новости.

Бот для управления тикетами:

1) тикеты, включает в себя ID тикета, пользователя, модератора, а также сообщения и даты создания и обновления;

2) пользователи, создатели тикетов и модераторы;

3) статусы тикетов, которые могут быть как открыты, закрыты или быть в работе.

2.1.2. Используемые классификаторы и системы кодирования

Для стандартизации данных в системе используются классификаторы и системы кодирования:

Коды статусов новостей:

1) 0 – ожидает проверки;

2) 1 – одобрена;

3) 2 – отклонена.

Коды статусов тикетов:

1) 0 – открыт;
2) 1 – в работе;
3) 2 – закрыт.

Коды типов новостных файлов:

1) 0 – текст;
2) 1 – фото;
3) 2 – видео;
4) 3 – документ;

2.1.3. Характеристика первичных документов с нормативно-справочной и входной оперативной информацией

Первичные документы включают данные, которые вводятся пользователями и модераторами:

Для модератора новостей это в первую очередь, сообщения пользователей с предложениями новостей, рисунок 21, а также решения модераторов (одобрение, отклонение, редактирование), рисунок 22.

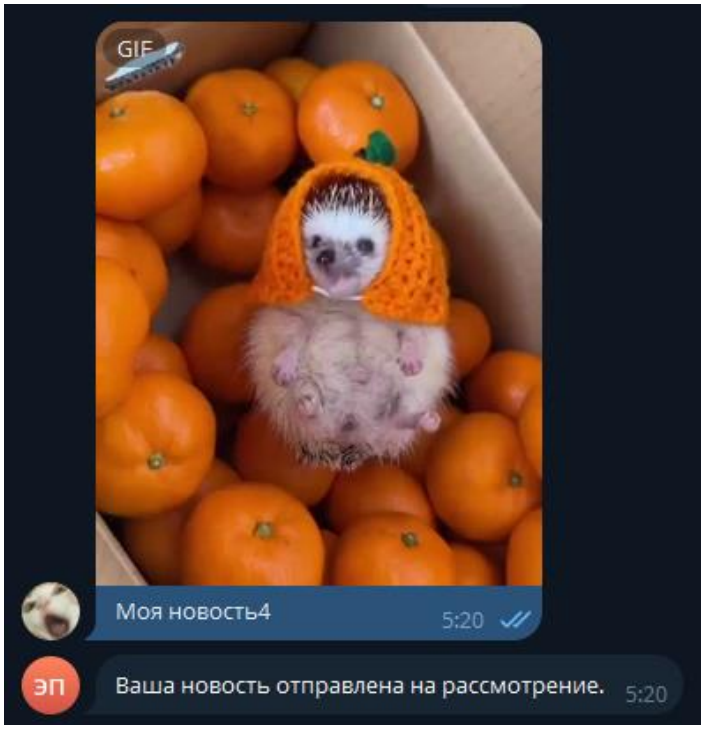

Рисунок 21 - Сообщение пользователя

Рисунок 22 - Действия для модераторов

Для бота – парсера новостей:

– RSS-каналы и данные из них (заголовки, ссылки, описания, даты публикации).

Для бота для управления тикетами:

– сообщения пользователей о проблемах или запросах, рисунок 23;

– ответы и действия модераторов, рисунок 24.

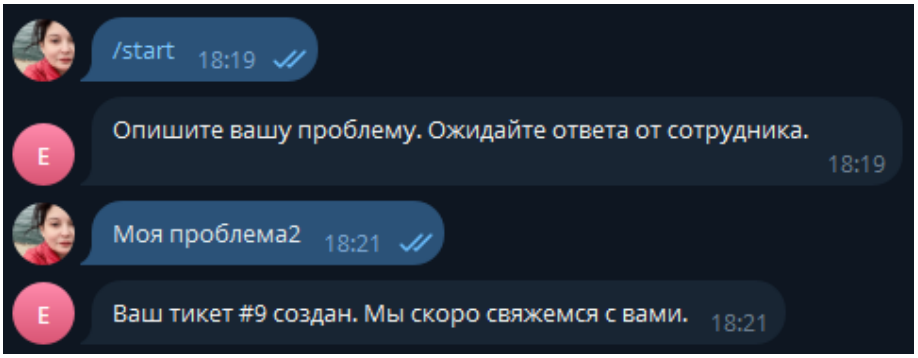

Рисунок 23 - Сообщение пользователя

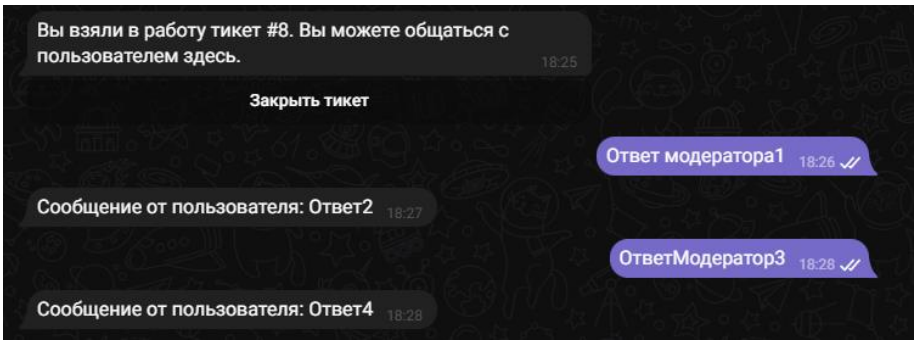

Рисунок 24 - Ответы и действия модераторов

2.1.4. Характеристика базы данных

База данных включает в себя несколько таблиц, каждая из которых отвечает за хранение определенных типов данных, таблицы 3, 4 и 5.

Таблица 3 – Таблица новостей

Обозначение	id	title	content	status	user_id	date_created
Тип данных	integer	text	text	integer	integer	text

Таблица 4 – Таблица источников новостей

Обозначение	id	source_url
Тип данных	integer	text

Таблица 5 – Таблица тикетов

Обозначение	id	user_id	moderator_ id	status	messages	date_created	date_updated
Тип данных	integer	integer	integer	integer	text	text	text

2.1.5. Характеристика результатной информации

Для модератора новостей, представляется в виде списка новостей с текущими статусами, рисунок 25, и уведомлением о новых предложенных записях, рисунок 26.

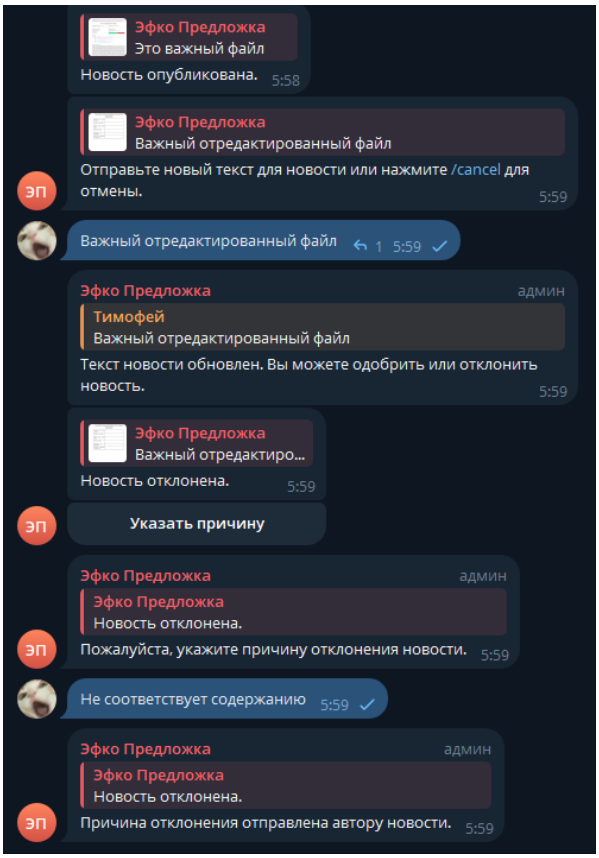

Рисунок 25 - Список новостей со статусами

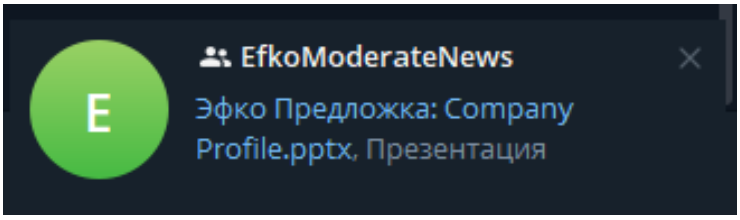

Рисунок 26 - Уведомление о предложенной записи

Для бота-парсера новостей, представляет собой публикации новостей в канале, рисунок 27, и отчёты о выполнении парсинга и публикации новостей, рисунок 28.

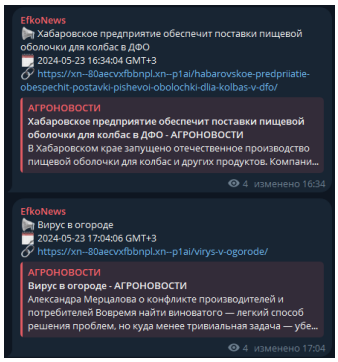

Рисунок 27 - Публикации в канале

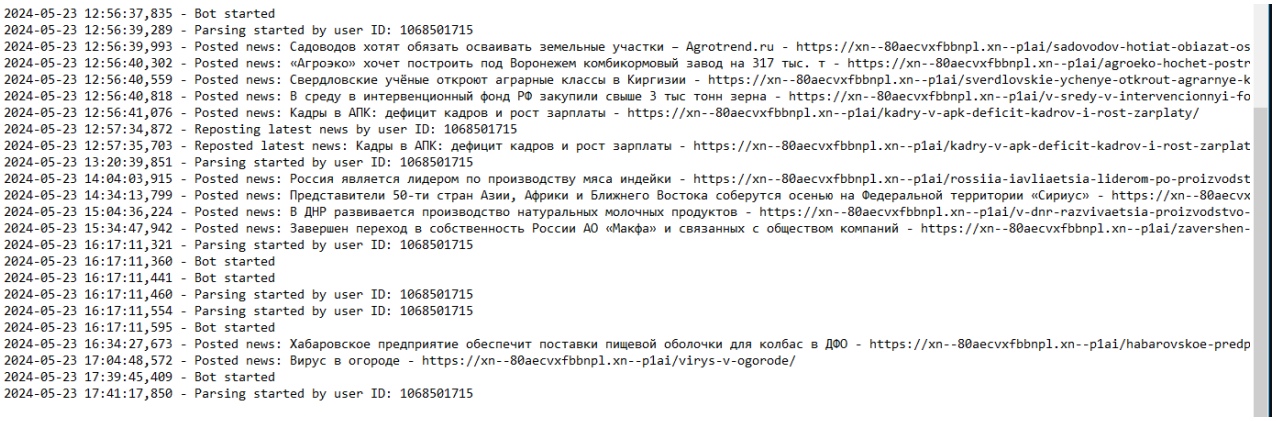

Рисунок 28 - Отчёт

Для бота для управления тикетами, существует список тикетов с информацией о каждом, рисунок 29, и уведомления о статусах тикетов и сообщениях, рисунок 30.

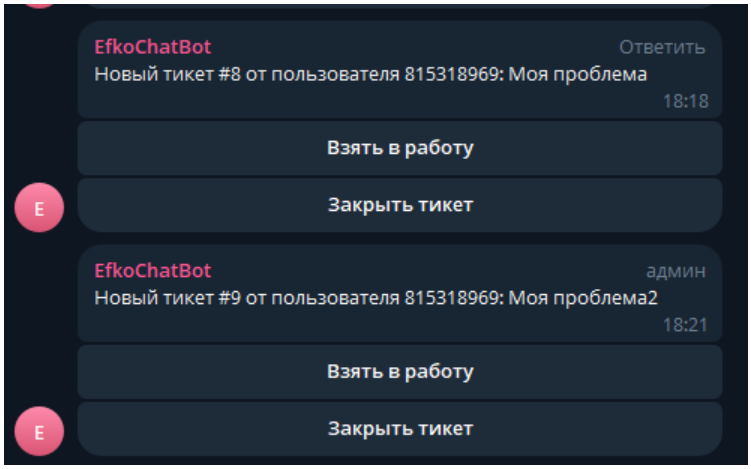

Рисунок 29 - Список существующих тикетов

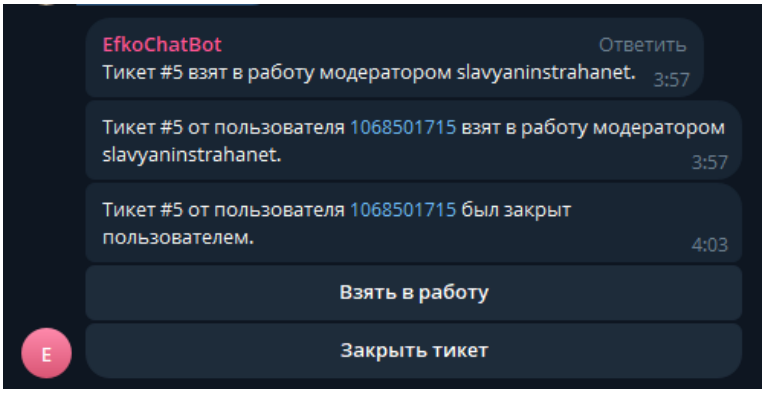

Рисунок 30 - Уведомление о статусе тикета

2.2. Программное обеспечение задачи

2.2.1. Общие положения

Бот модератор новостей, позволяет пользователям отправлять новости, блок-схема на рисунке 31, а также позволяет модераторам редактировать новости, блок-схема на рисунке 32.

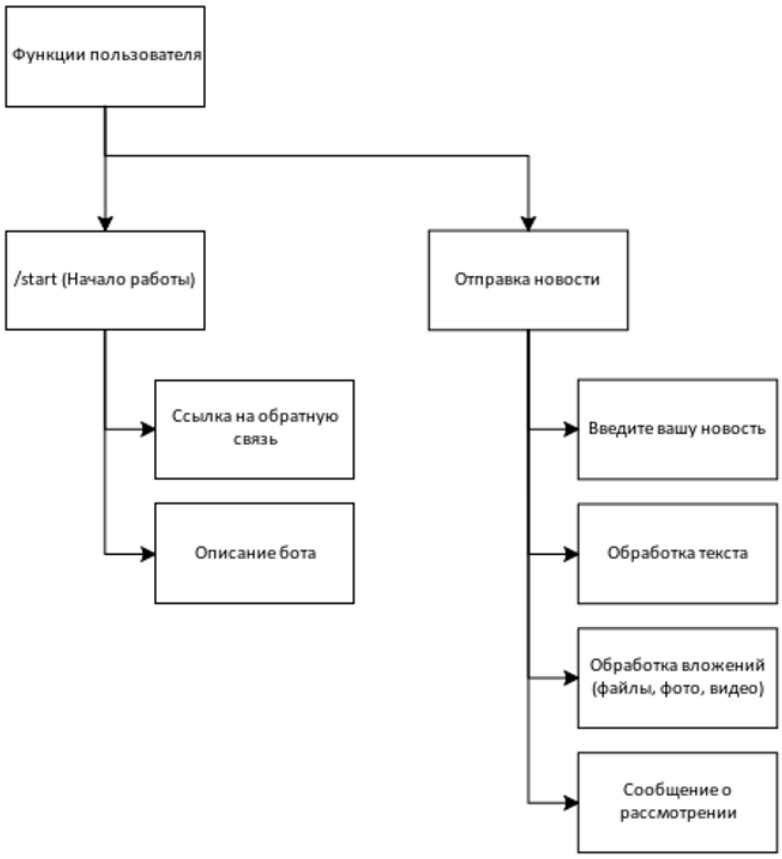

Рисунок 31 - Блок-схема для пользователя

Бот-парсер новостей, позволяет модерации парсить новости из источников и автоматически публиковать их в канал. Бот поддерживает добавление модерации. Блок-схема на рисунке 33.

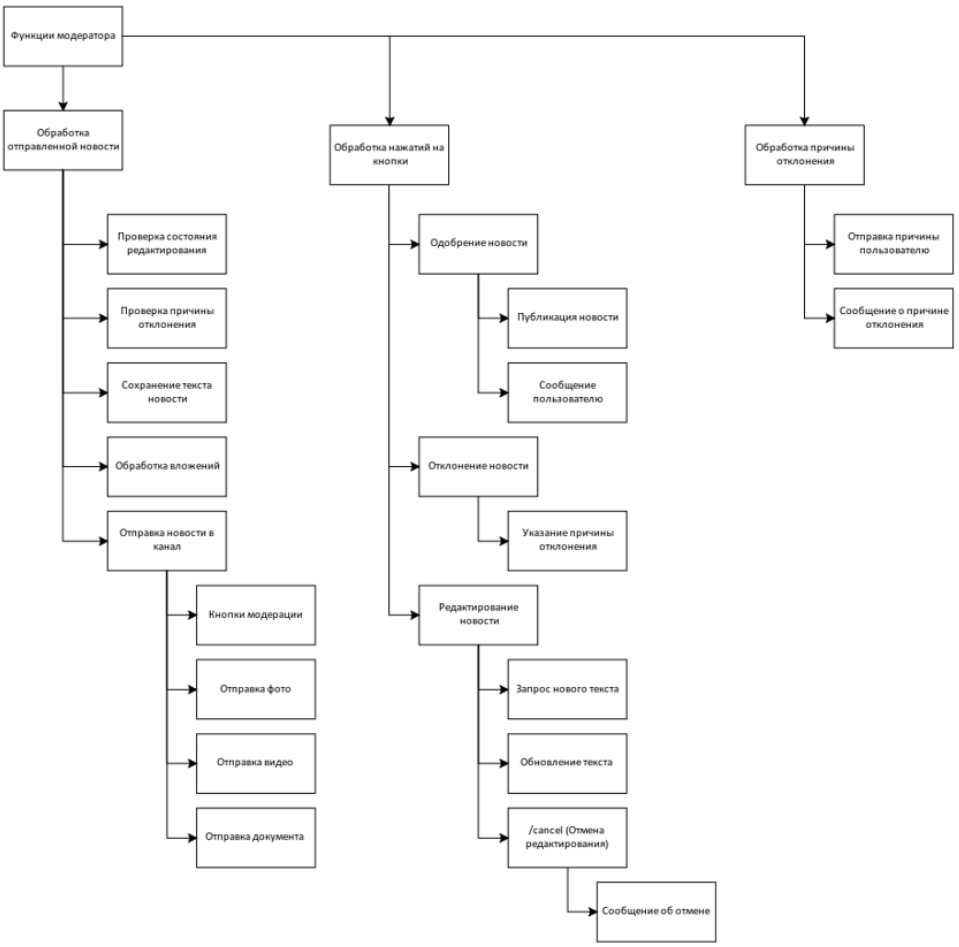

Рисунок 32 - Блок-схема для модератора

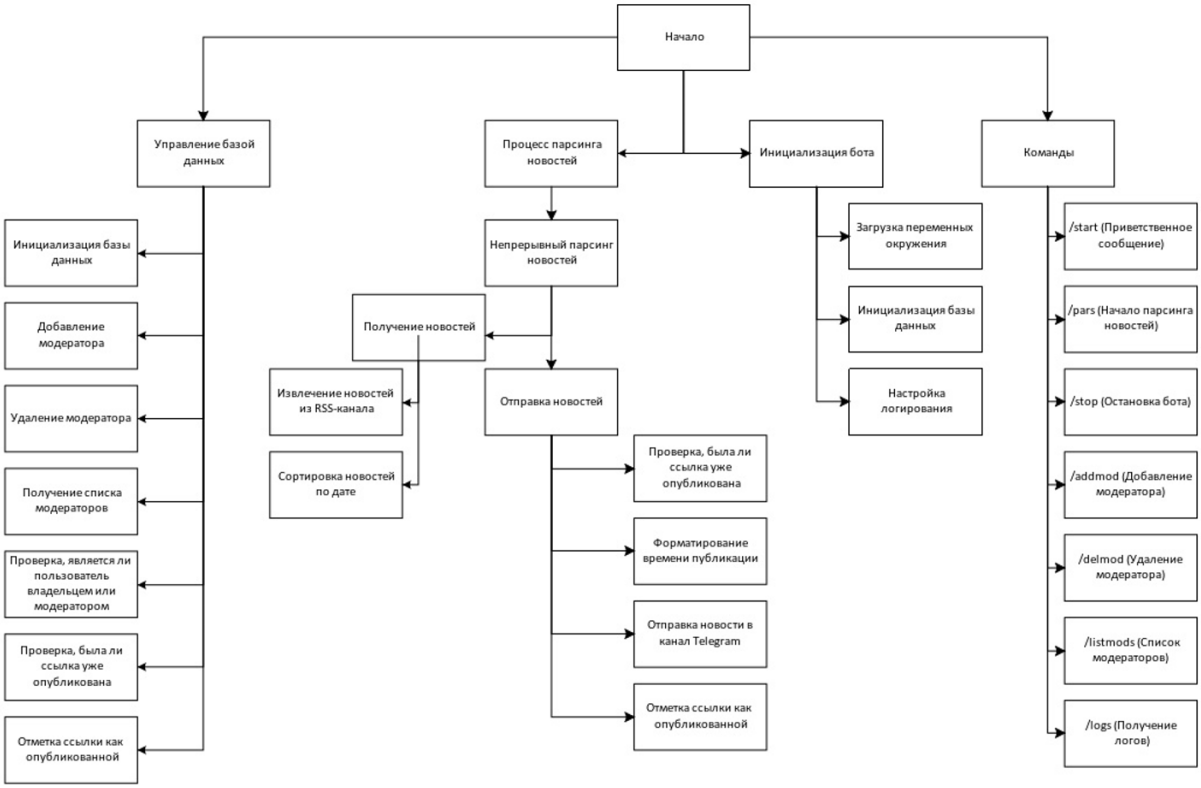

Рисунок 33 - Блок-схема бота-парсера

Бот для управления тикетами или чат-бот, представлен для обратной связи с модерацией, блок-схема на рисунке 34.

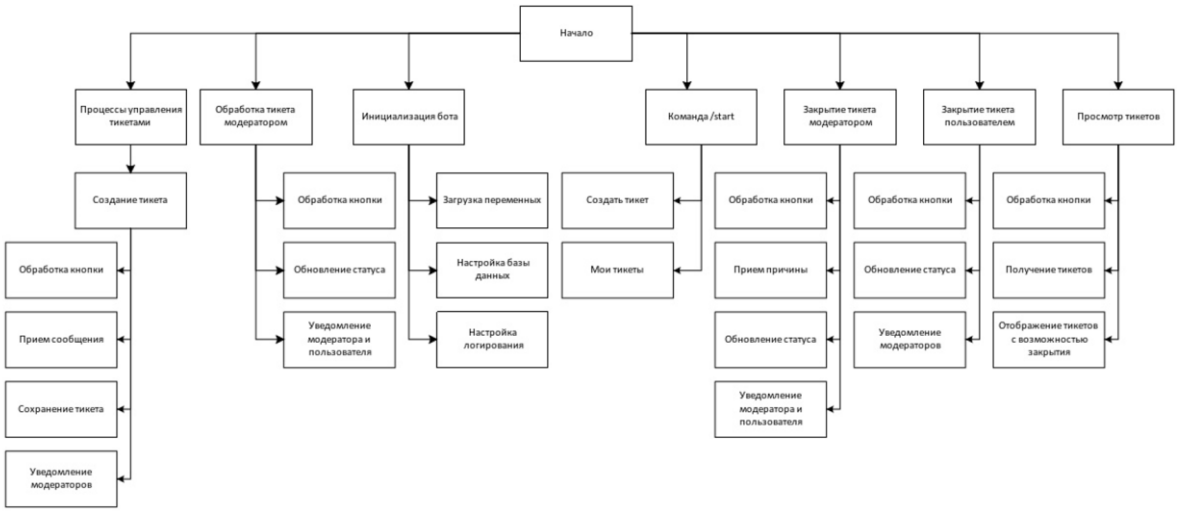

Рисунок 34 - Блок-схема бота для управления тикетами

Сценарии диалога представлены в приложении. Они описывают основные шаги взаимодействия в диалогах для каждой из программы.

Приложение А сценарий для бота парсинга новостей. В нём пользователь взаимодействует с ботом, введя команду /start и /pars, чтобы запустить парс новостей для публикации.

Приложение Б сценарий для бота управления тикетами. Пользователь вводит команду /start и создаёт тикет для дальнейшего общения с модерацией.

Приложение В сценарий для бота отправки новостей. В этом сценарии пользователь вводит /start, отправляет новость и ожидает ответа от модератора.

Все эти сценарии представляют базовые функции ботов.

2.2.2. Структура программных модулей

Структура программных модулей представлена на блок-схеме рисунка 35. На ней изображены все 3 бота, начиная с модуля для работы с API Telegram, «Создание и взаимодействие с мессенджером telegram осуществляется с помощью Telegram API Bot, которая включает в

себяобъекты и команды, предназначенные для установки поведения бота.» [11].

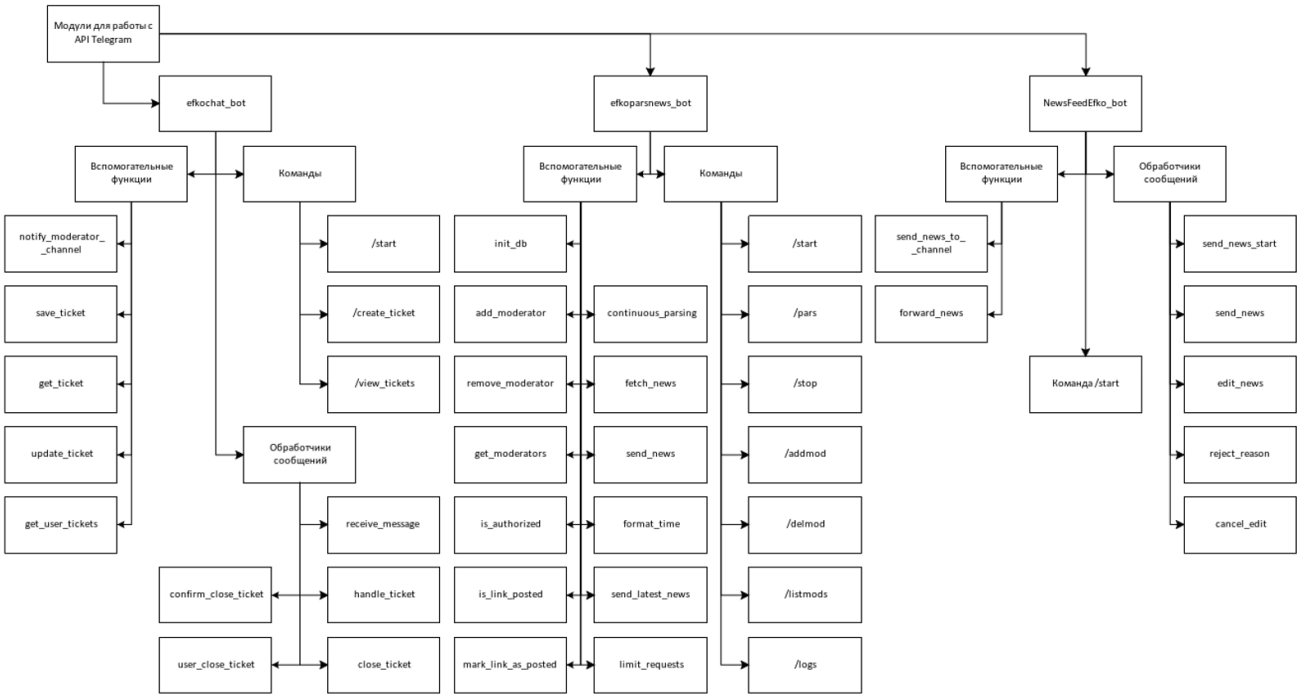

Рисунок 35 - Структура программных модулей

2.2.3. Описание программных модулей

Модуль efkochat_bot.

Код бота, в приложении А. Разбор ключевых методов реализации:

– импорт библиотек и модулей: logging, os, json, sqlite3, dotenv, telegram и telegram.ext;

– загрузка переменных окружения: TELEGRAM_BOT_TOKEN, OWNER_ID и MODERATOR_CHANNEL_ID из файла .env.

Команды:

– /start – начало взаимодействия с ботом;

– create_ticket – начало создания нового тикета;

– view_tickets – просмотр всех тикетов пользователя.

Обработчики сообщений:

– receive_message – прием и обработка сообщений от пользователей и модераторов;

– handle_ticket – обработка тикета модератором;

– close_ticket – инициирует закрытие тикета модератором;

– confirm_close_ticket – подтверждение закрытия тикета без причины;

– user_close_ticket – закрытие тикета пользователем.

Вспомогательные функции:

– notify_moderator_channel – уведомление канала модераторов о новом тикете;

– save_ticket – сохранение тикета в базу данных;

– get_ticket – получение информации о тикете по его ID;

– update_ticket – обновление информации о тикете;

– get_user_tickets – получение всех тикетов пользователя.

Алгоритм создания тикета, на блок-схеме на рисунке 36.

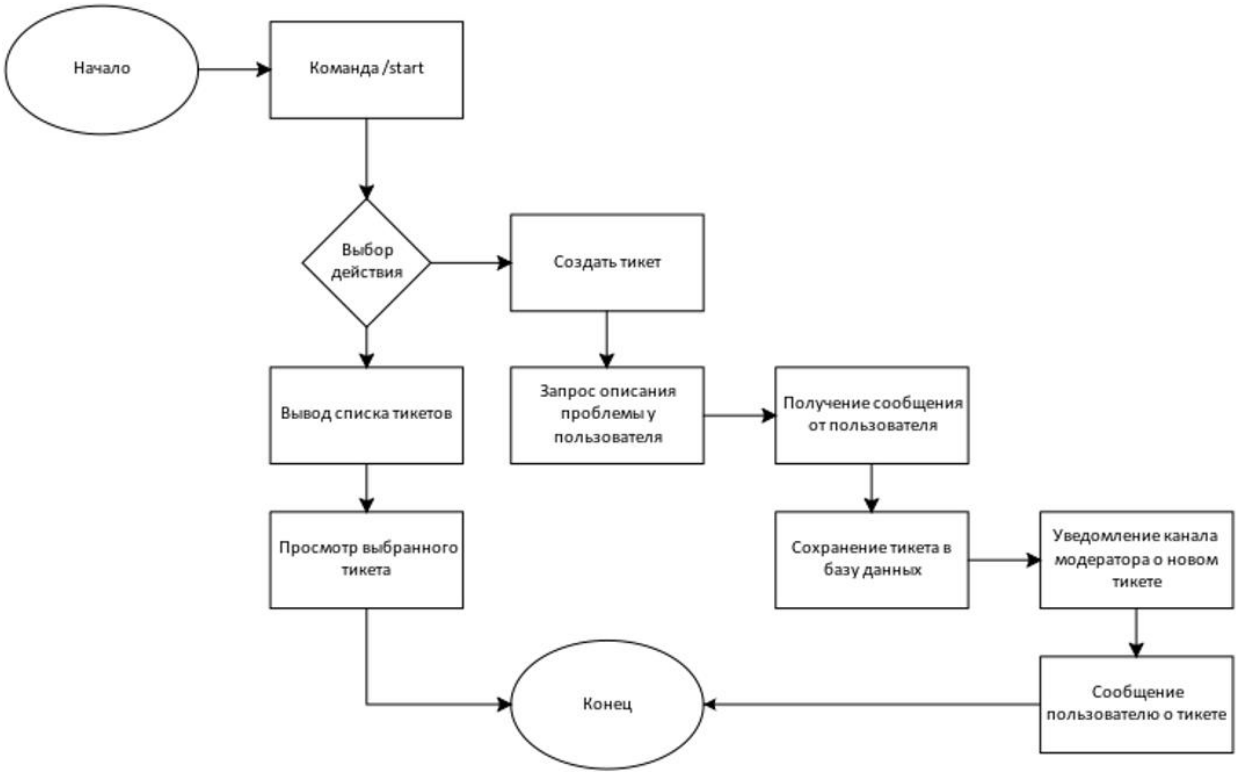

Рисунок 36 - Блок-схема алгоритм создания тикета

Модуль efkoparsnews_bot.

Код бота, в приложении Б. Разбор ключевых методов реализации:

– импорт библиотек: Сценарий использует библиотеки os, feedparser, telebot, sqlite3, logging, threading, time, bs4 (BeautifulSoup), dotenv, datetime, functools, и dateutil;

– загрузка переменных окружения: Загружаются переменные из файла TOKEN.env, включая токен бота, ID чата и ID владельца;

Команды:

– /start – начало работы с ботом;

– /pars – начало парсинга новостей;

– /stop – остановка бота;

– /addmod – добавление модератора;

– /delmod – удаление модератора;

– /listmods – просмотр списка модераторов;

– /logs – просмотр логов.

Вспомогательные функции:

– init_db – инициализация базы данных;

– add_moderator – добавление модератора в базу данных;

– remove_moderator – удаление модератора из базы данных;

– get_moderators – получение списка модераторов;

– is_authorized – проверка авторизации пользователя;

– is_link_posted – проверка, была ли ссылка уже опубликована;

– mark_link_as_posted – отметка ссылки как опубликованной;

– fetch_news – получение новостей из RSS-канала;

– format_time – форматирование времени публикации;

– send_news – отправка новости в Telegram канал;

– send_latest_news – отправка самой последней новости в Telegram канал;

– continuous_parsing - непрерывный парсинг новостей;

– limit_requests - ограничение на количество запросов (антиспам).

Алгоритм для парсинга новостей, на блок-схеме, рисунок 37.

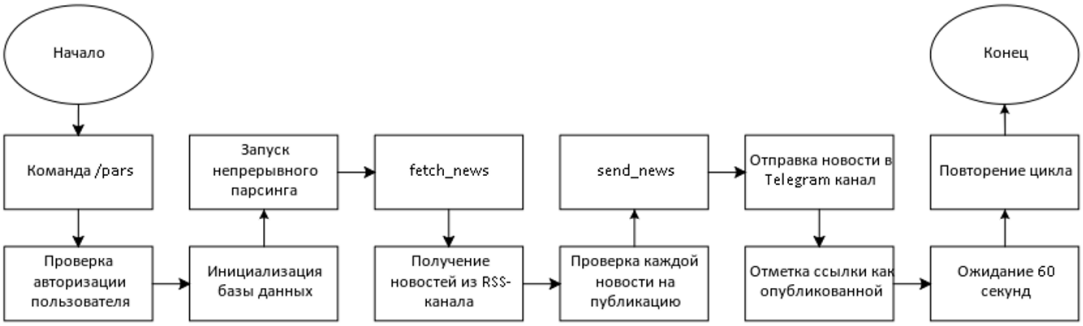

Рисунок 37 - Блок-схема парсинга новостей

Модуль NewsFeedEfko_bot.

Код бота, в приложении В. Разбор ключевых методов реализации:

Импорт необходимых библиотек и модулей:

– os для работы с переменными окружения;

– load_dotenv из библиотеки dotenv для загрузки переменных из файла .env.

Команды:

– /start – начало работы с ботом.

Обработчики сообщений:

– send_news_start – начало отправки новости пользователем;

– send_news – обработка отправленной новости;

– edit_news – обработка редактирования новости;

– reject_reason – обработка причины отклонения новости;

– cancel_edit – отмена редактирования новости.

Вспомогательные функции:

– send_news_to_channel – отправка новости в канал;

– forward_news – пересылка новости в канал.

Алгоритм для отправки новостей на блок-схеме, рисунок 38.

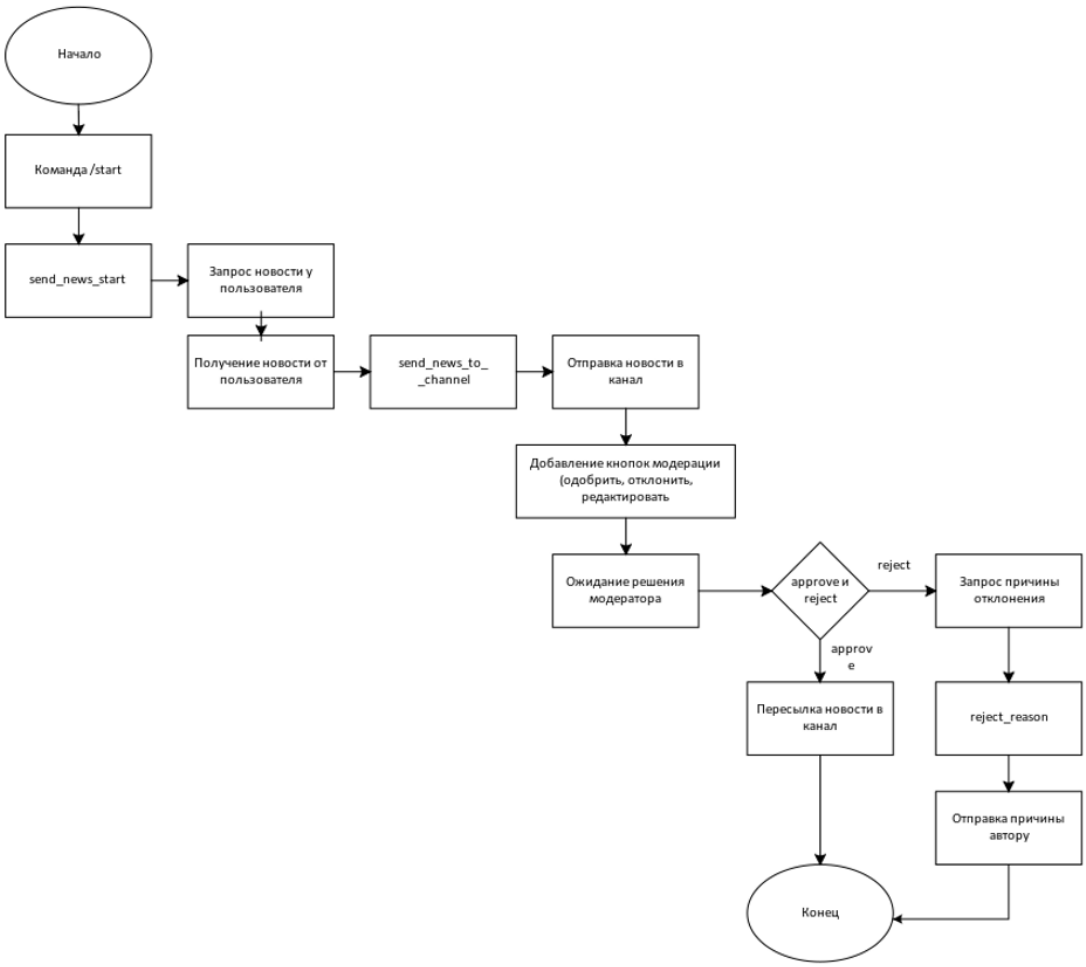

Рисунок 38 - Блок-схема отправка новостей

2.3. Технологическое обеспечение задачи

2.3.1. Первичная настройка ботов с помощью BotFather

В самом приложении Telegram используется BotFather, это официальный бот в мессенджере, с помощью которого можно создавать и управлять другими ботами. Он предоставляет функционал для регистрации новых ботов, получения токенов доступа, настройки информации о ботах и управления их поведением. «Благодаря удобному интерфейсу и типовому набору команд BotFather позволяет выполнить операцию создания бота посредством диалога, как это всегда происходит при общении с обычным пользователем.». [2]

С помощью команды /newbot создаются все необходимые боты, рисунок 39,

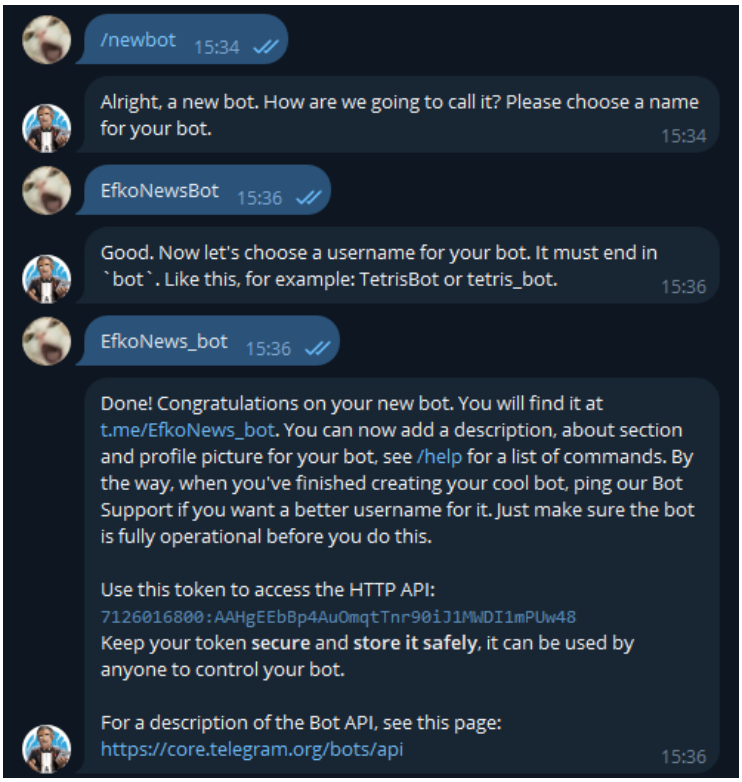

Рисунок 39 – Создание нового бота

Командой /mybots можно увидеть список созданных ботов, рисунок 40.

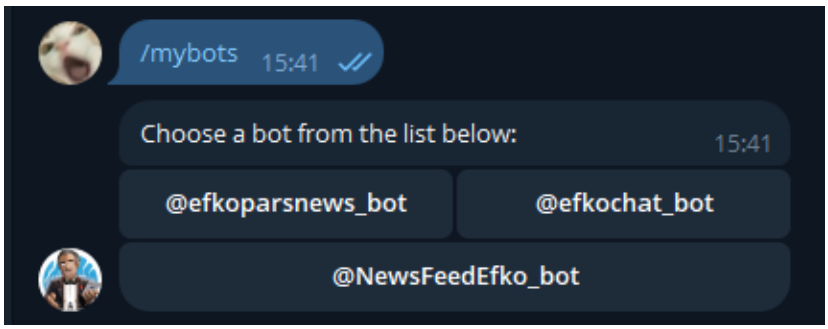

Рисунок 40 – Список всех ботов

2.3.2. Дополнительные настройки

Каждый бот нуждается в дополнительной настройке прав и редактировании описания, рисунки 41 – 46.

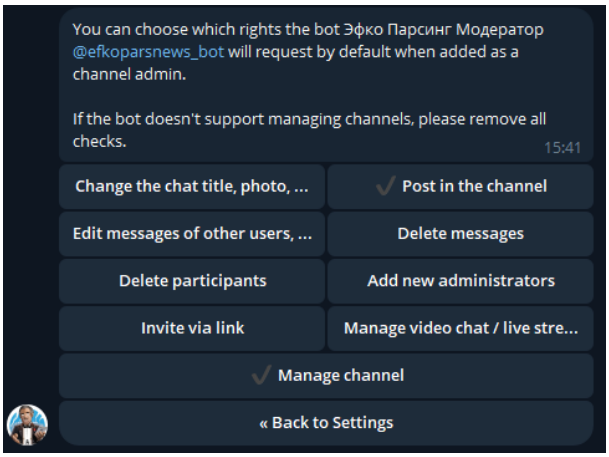

Рисунок 41 – Выставление необходимых настроек

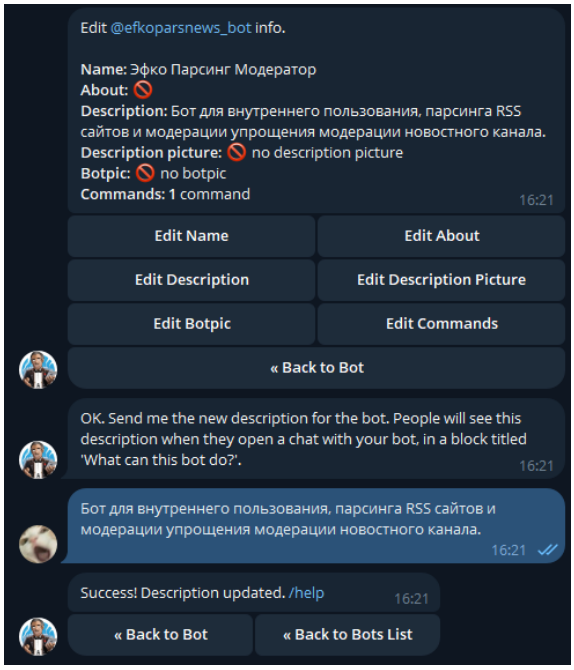

Рисунок 42 – Изменение описания

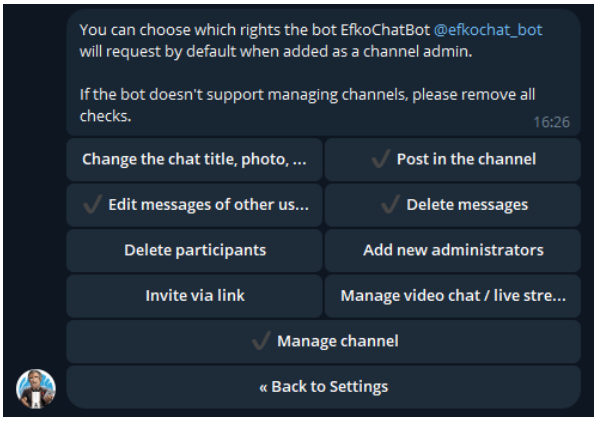

Рисунок 43 – Права для чат бота

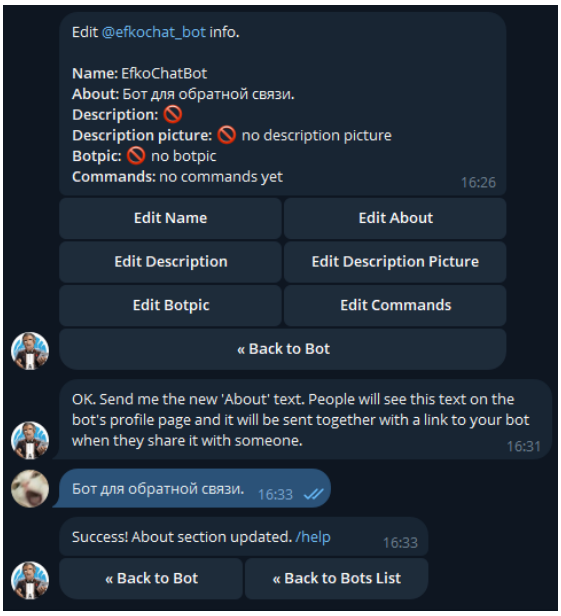

Рисунок 44 – Изменение информации

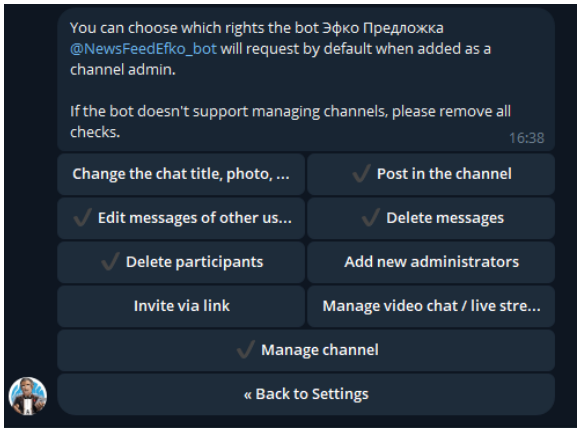

Рисунок 45 – Добавление необходимых прав для бота

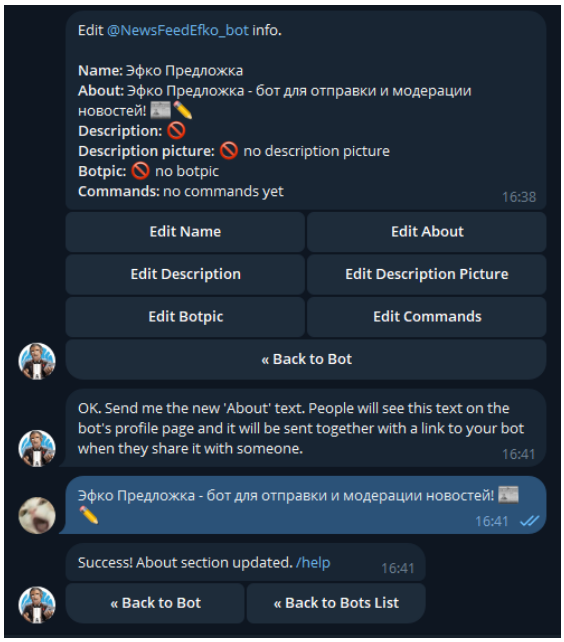

Рисунок 46 – Редактирование информации

2.3.3. Схема технологического процесса сбора, передачи, обработки и выдачи информации

Сбор информации осуществляется через несколько источников: пользователи бота отправляют новости и RSS-каналы предоставляют новости, блок-схема на рисунке 47.

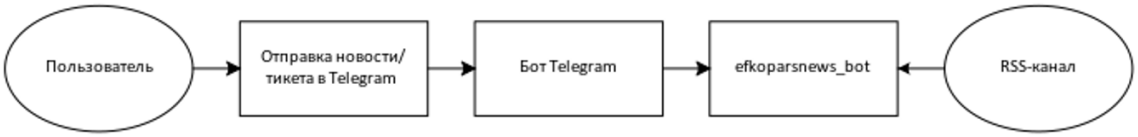

Рисунок 47 - Блок-схема процесса сбора информации

Процесс передачи информации включает отправку новостей пользователями, пересылку новостей в канал модерации и уведомление о новых тикетах, схема на рисунке 48.

Рисунок 48 - Схема процесса передачи информации

Процесс выдачи информации, заключается в отправке одобренных, промодерированных, уведомление пользователя и публикация новостей из RSS-канала, схема на рисунке 49.

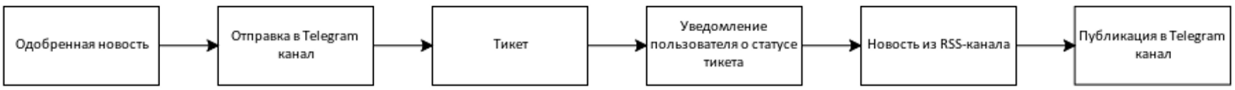

Рисунок 49 - Схема процесса выдачи информации

Процесс обработки информации осуществляется с помощью модерации новостей модераторами, управления тикетами и проверками новостей из RSS-каналов, блок-схема алгоритма на рисунке 50.

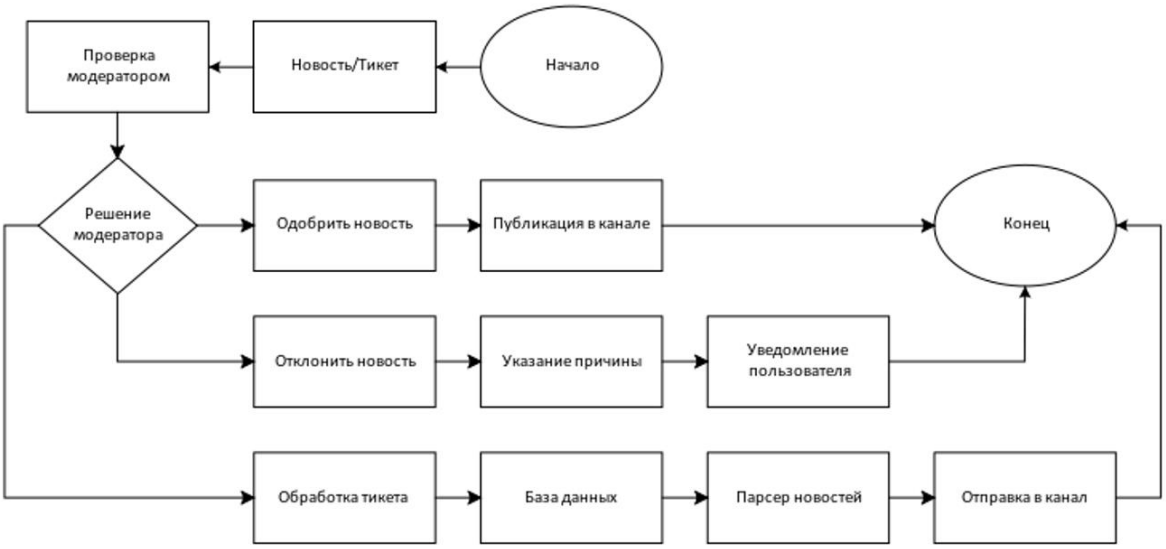

Рисунок 50 – Блок-схема процесс обработки информации

Все эти процесс обеспечивают полноценный цикл работы ботов, начиная от сбора, обработки и выдачи информации аудитории канала.

2.4. Руководство пользователя

2.4.1. Руководство пользователя для efkoparsnews_bot

Запуск бота, рисунок 51.

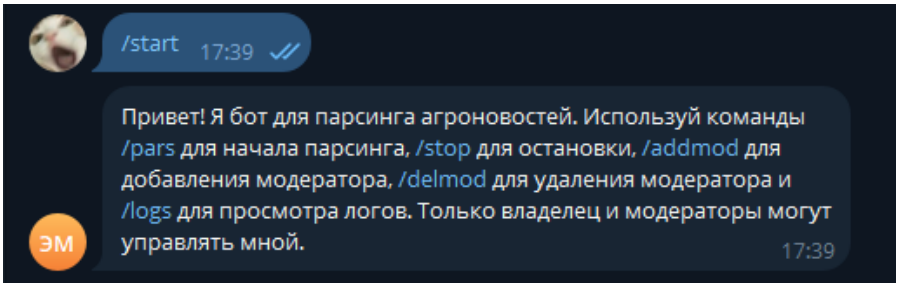

Рисунок 51 – Запуск бота

Использование команды /pars для парсинга новостей, рисунок 52.

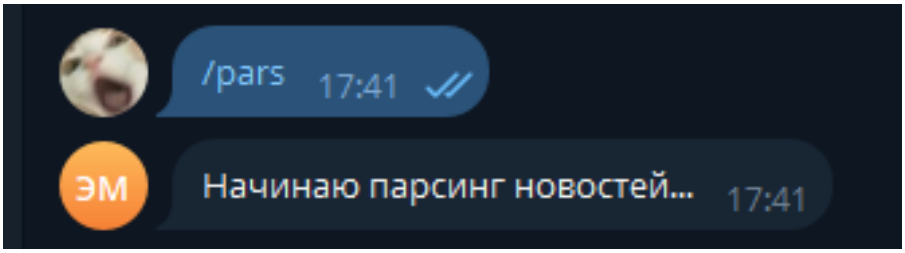

Рисунок 52 – Парсинг новостей

Бот запущен, когда на сайте появиться новая новость, она сразу же появиться в канале. С помощью даты в посте, даты отправки сообщения в канал и даты на сайте, мы можем удостовериться в том, что бот работает исправно, рисунок 53 и 54. Посты были отредактированы для компактности, пример поста без редактирования, на рисунке 55.

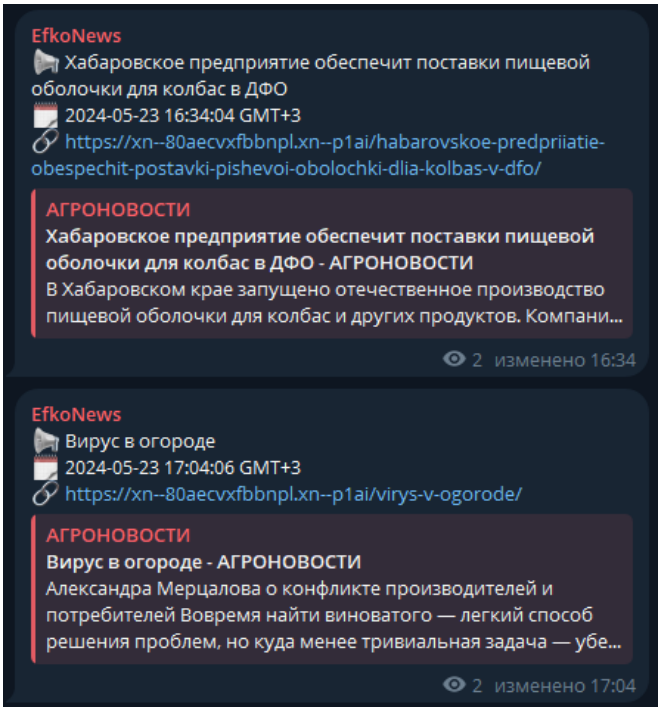

Рисунок 53 – Посты в канале

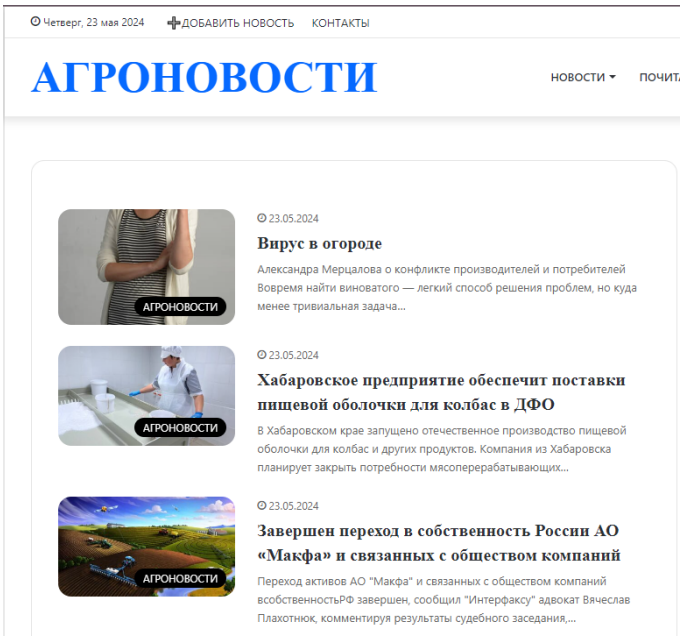

Рисунок 54 – Посты на сайте

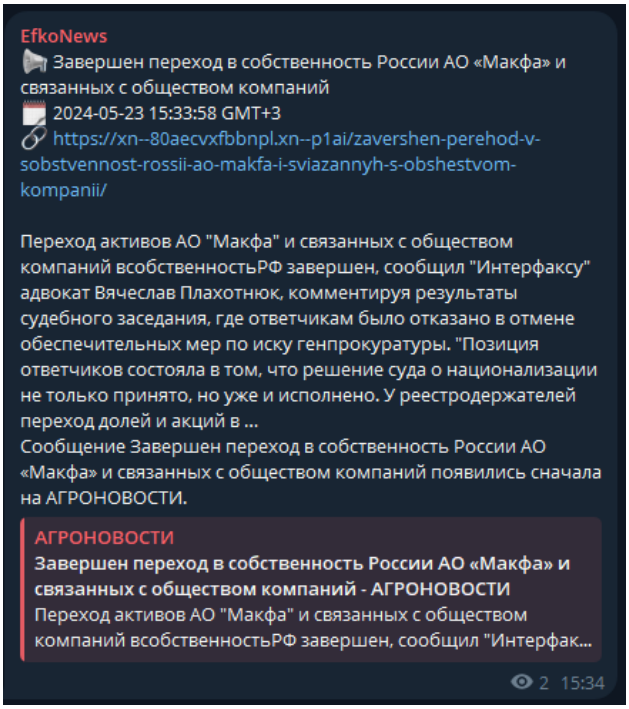

Рисунок 55 – Пост без редактирования

Для выявления проблем есть возможность посмотреть логи работы программы, рисунок 56.

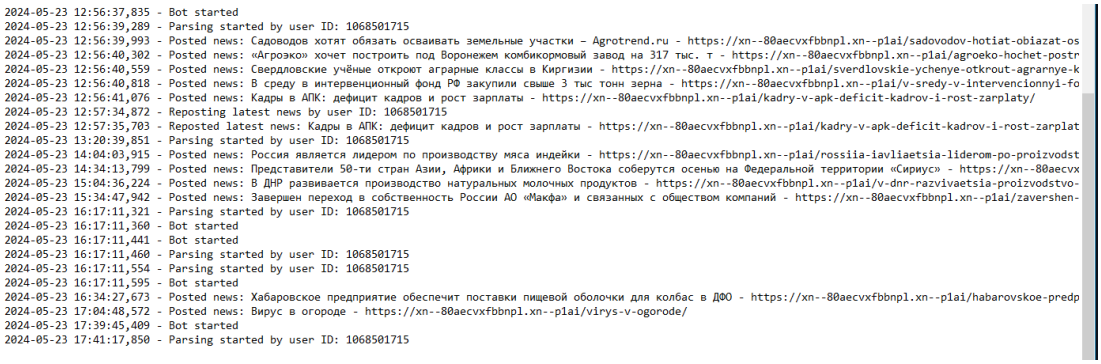

Рисунок 56 – Лог в текстовом редакторе

С другого Telegram аккаунта используем любую из команд, рисунок 57.

Вернёмся на аккаунт владельца и добавим этого пользователя в модераторы, указав ID этого пользователя, рисунок 56.

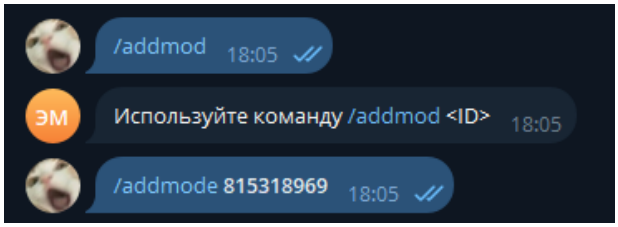

Рисунок 56 – Добавление модератора

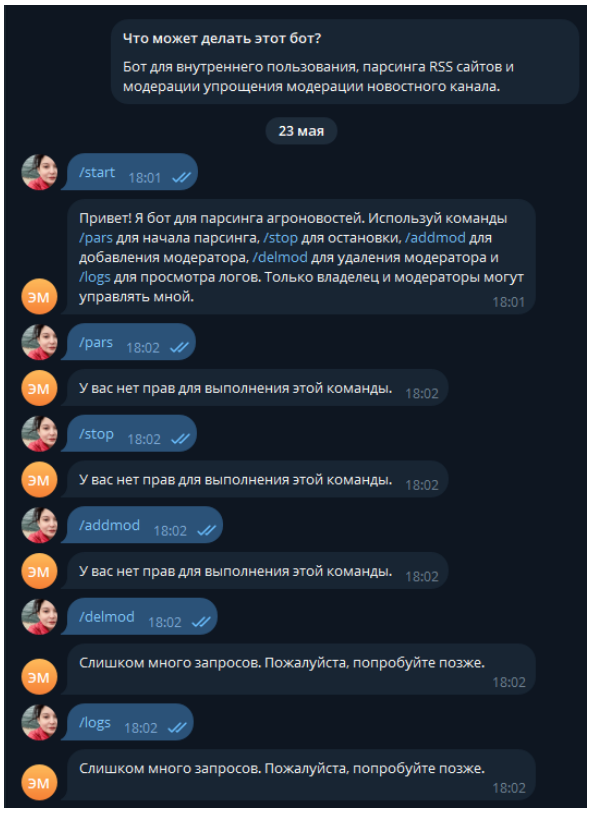

Рисунок 57 – Использование команд с другого аккаунта

Попробуем использовать команду с другого аккаунта, рисунок 58, а после удалим модератора, рисунок 59, и попробуем опять использовать команду, рисунок 60.

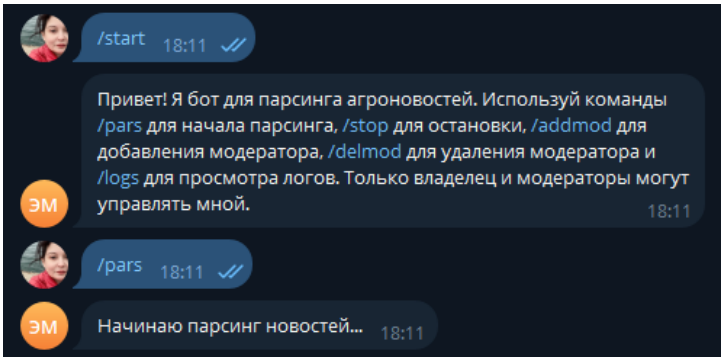

Рисунок 58 – Модератор добавлен успешно

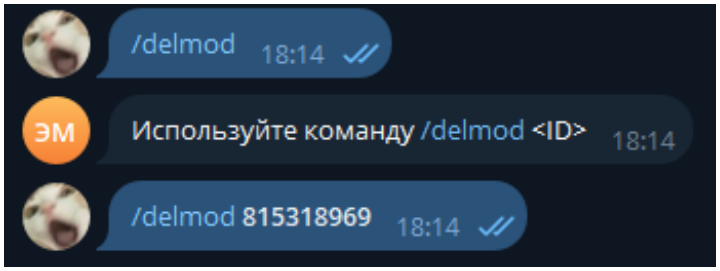

Рисунок 59 – Модератор удалён

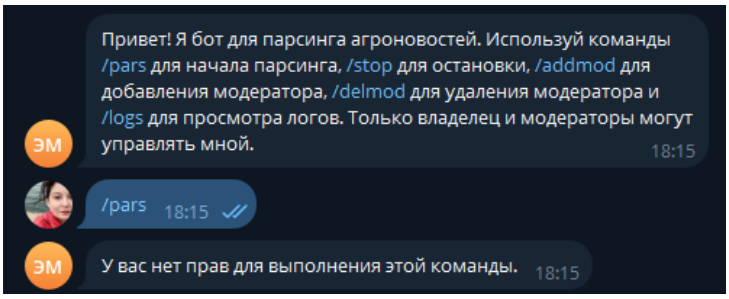

Рисунок 60 – Отсутствие доступа

2.4.2. Руководство пользователя для efkochat_bot

Запуск бота, рисунок 61.

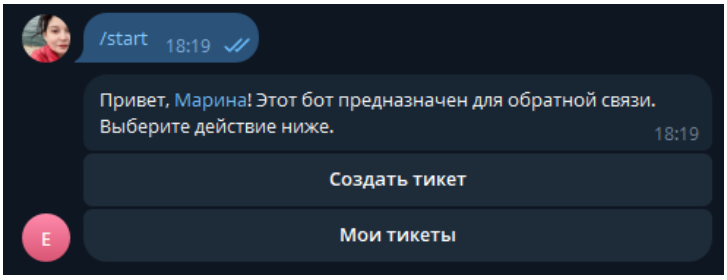

Рисунок 61 – Запуск бота

Создадим новый тикет нажав на соответствующую кнопку, опишем проблему, после создания просмотрим все тикеты, которые создавал этот пользователь, рисунок 62.

Рисунок 62 – Создание тикета и мои тикеты

В чат модераторов поступило два тикета от пользователя. Тикет 8 возьмём в работу и ответим на него от лица модератора и от лица пользователя, рисунок 63 и 64.

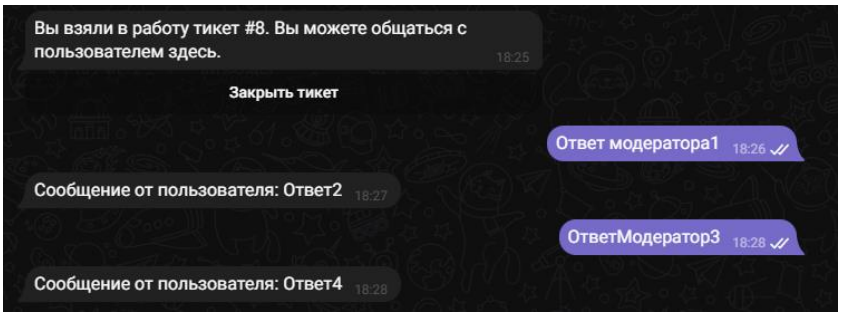

Рисунок 63 – Чат от лица модератора

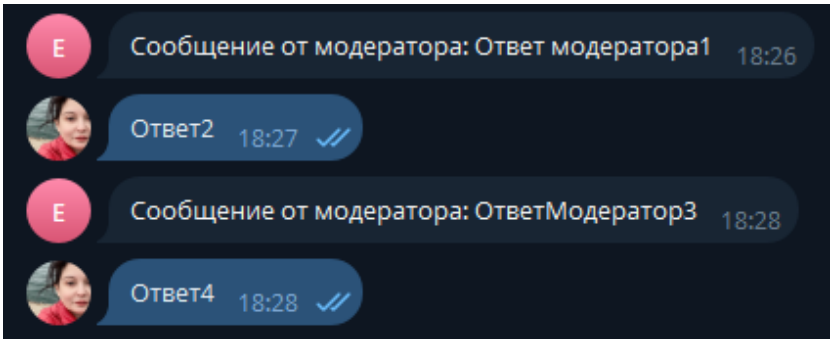

Рисунок 64 – Чат от лица пользователя

Закроем тикет. При закрытии пользователь получает соответствующее уведомление и статус тикета меняется на closed, рисунок 65.

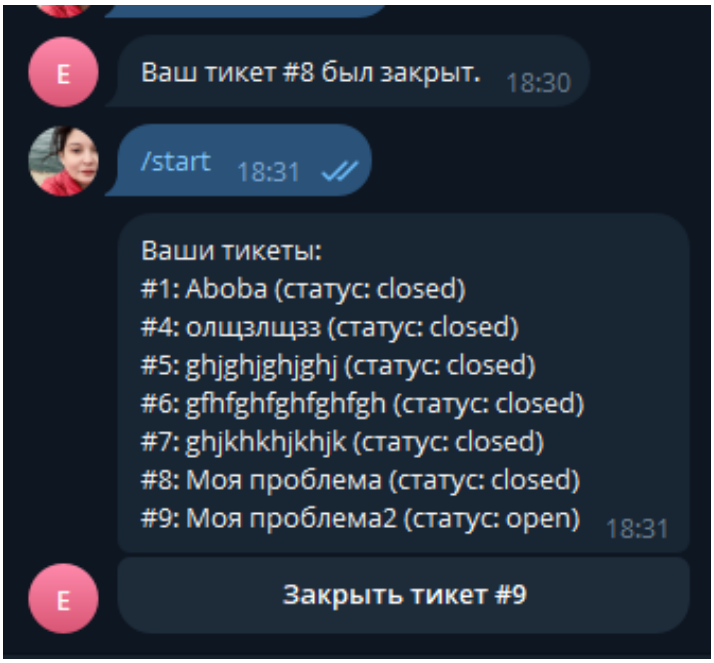

Рисунок 65 – Уведомление и проверка статуса тикета

От лица пользователя закроем последний тикет, рисунок 66. Нажимаем на соответствующую кнопку, получаем уведомление и проверяем статус тикета.

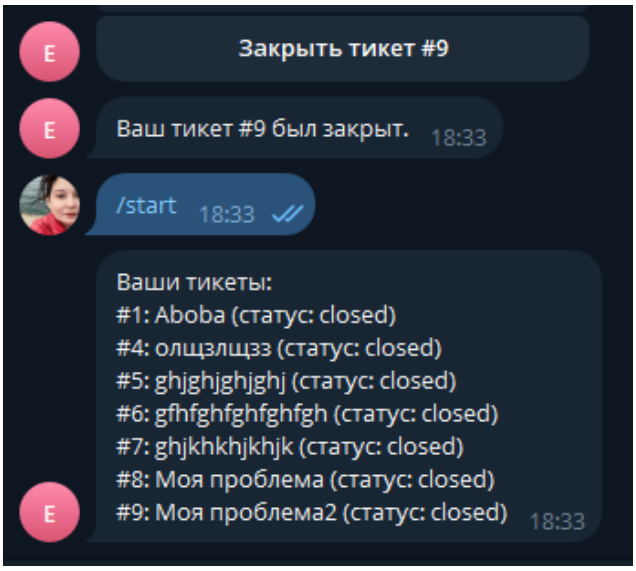

Рисунок 66 – Закрытие тикета пользователем

Приложение Г код программы для бота парсинга новостей. Приложение Д код программы для бота управления тикетами. Приложение Е код программы для бота отправки новостей.

2.4.3. Руководство пользователя для NewsFeedEfko_bot

Запуск бота и отправление нескольких новостей на рассмотрение, рисунок 67.

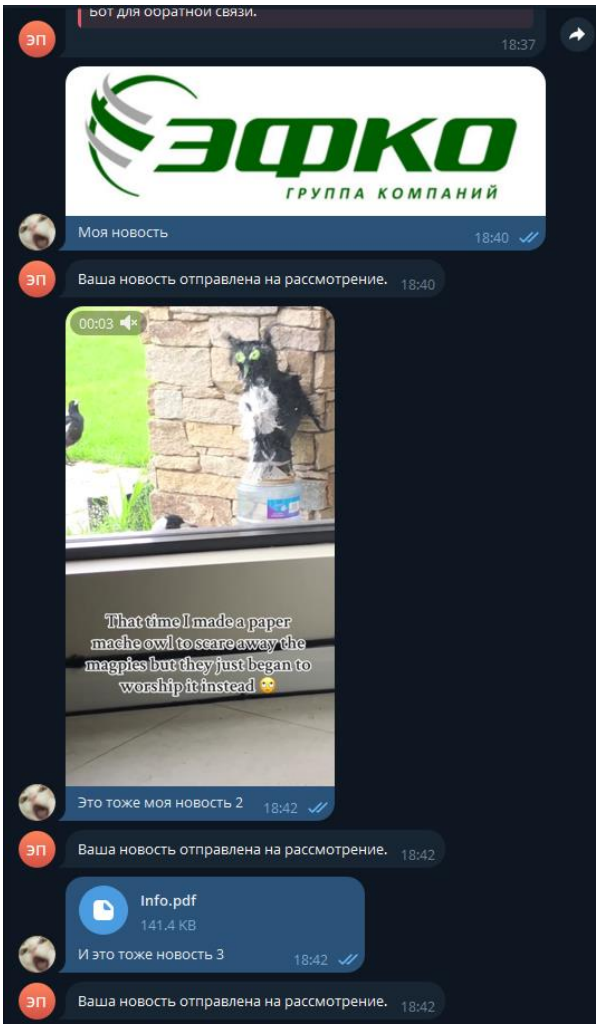

Рисунок 67 – Запуск бота и отправка новостей

В отдельном канале для модераторов, получаем эти новости. Первую одобрим, рисунок 68. Вторую отклоним, рисунок 69. Третью отредактируем и одобрим, рисунок 70.

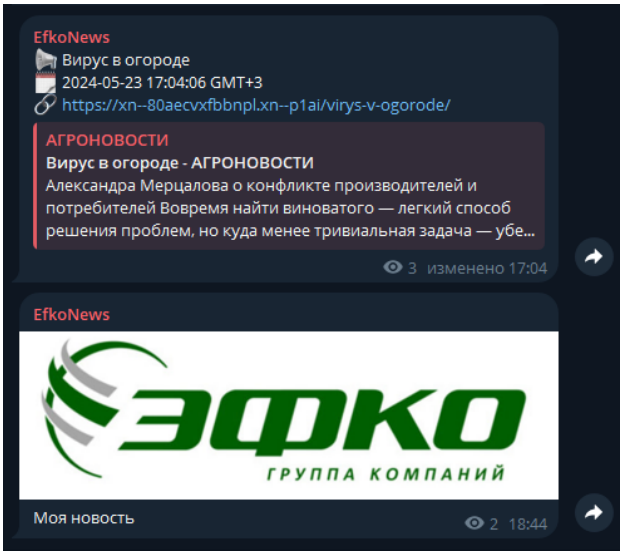

Рисунок 68 – Одобренная новость

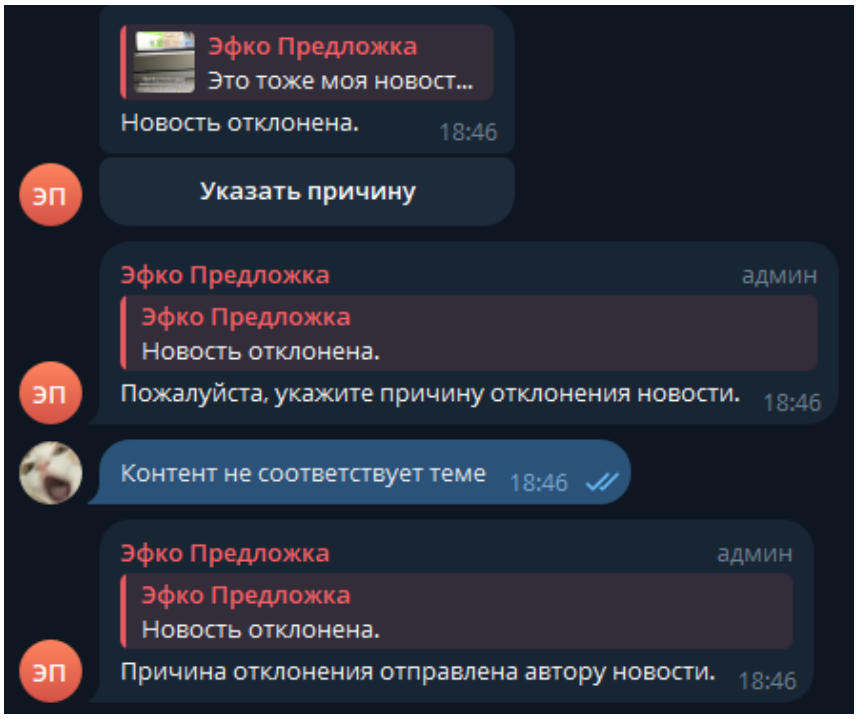

Рисунок 69 – Отклонение новости

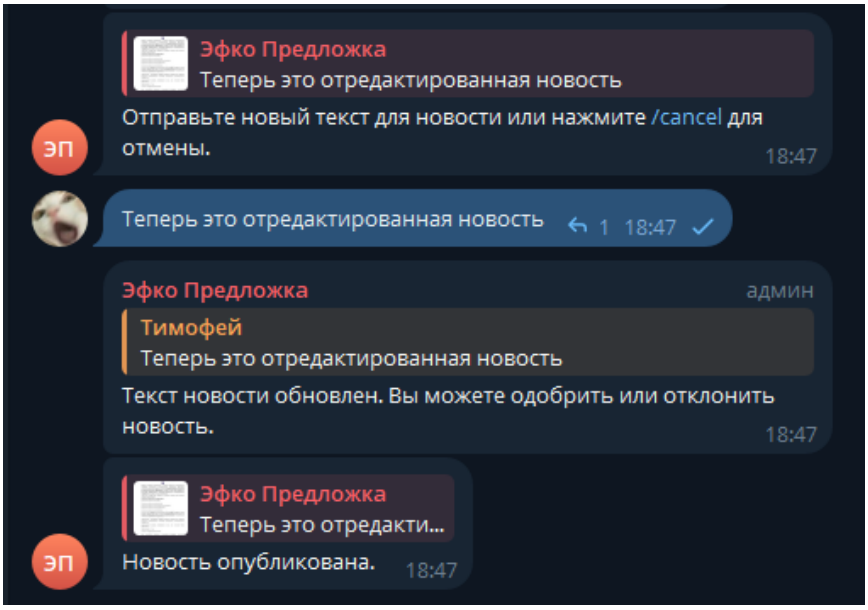

Рисунок 70 – Редактирование и публикация

При отклонении новости или публикации пользователь получает соответствующее уведомление, рисунок 71.

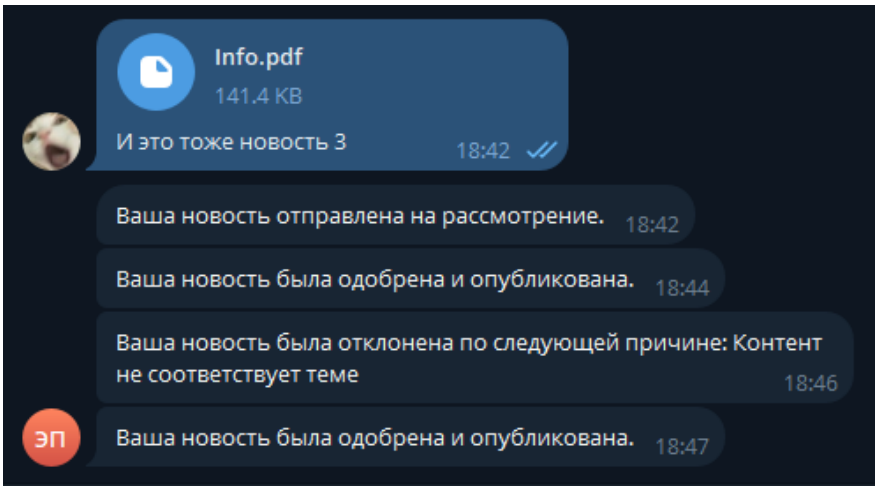

Рисунок 71 – Уведомления пользователя

Новостной канал, рисунок 72.

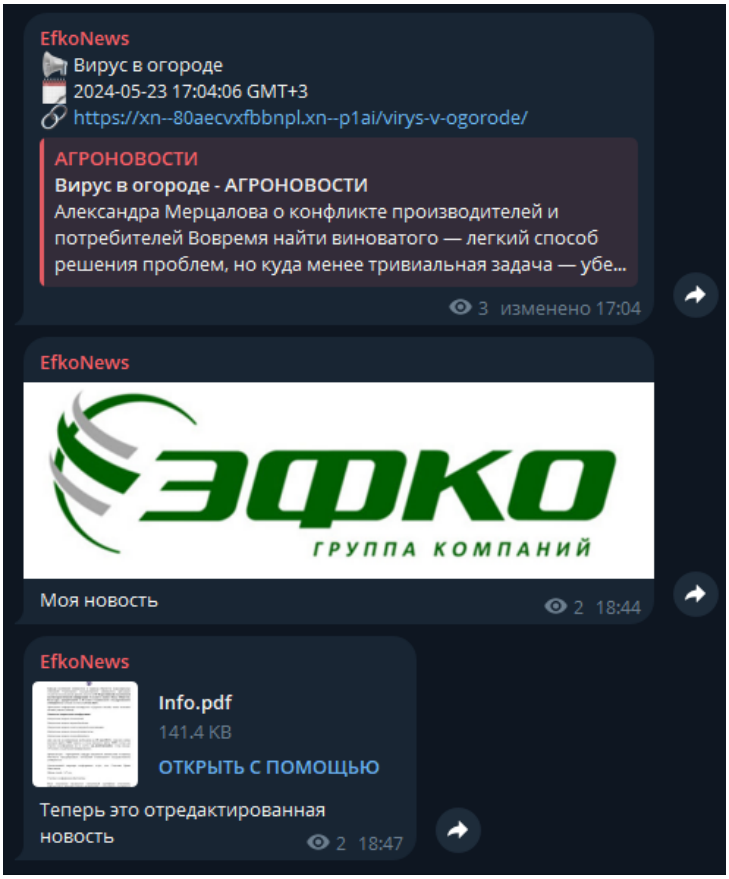

Рисунок 72 – Опубликованная и отредактированная новость

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были достигнута основная цель и выполнены задачи.

Целью исследования является, улучшение системы оповещения сотрудников. Задачами исследования является:

– анализ существующих решений и требований к функциональности модуля;

– обоснование выбора основных проектируемых решений;

– разработка модуля «Новостная лента»;

– тестирование и отладка разработанного модуля;

– обоснование экономической эффективности работы;

– оценка эффективности модуля.

Объектом исследования, будет являться мессенджеры как система управления корпоративными коммуникациями. Сами мессенджеры были выбраны, потому что являются максимально удобным для пользователя приложением, не представляющем лишние функции. Предметом исследования будет являться модуль «Новостная лента».

В качестве мессенджеры был выбран именно Telegram, потому что, он обладает большей и расширяющейся аудиторию, в отличии от всех остальных мессенджеров.

В качестве методов исследования, использовался контент-анализ, чтобы анализировать содержания текстовых и мультимедийные данные для выявления значимых тенденций и закономерностей. Для оценки потенциала внедрения новостного модуля, анализировался функционал, преимущества и недостатки программ. Также проанализировали распространённость и охват аудитории этих программ.

Работа включает в себя, исследовательский раздел, в котором проводиться анализ систем управления корпоративными коммуникациями, выбор систем управления корпоративными коммуникациями для реализации

новостной ленты и обзор аналогов новостных лент в систем управления корпоративными коммуникациями, характеристика и структура предприятия, и формализация расчетов подзадач.

Экономическая эффективность модуля подтверждается расчётами, согласно которым, проект начнет приносить чистую экономическую прибыль уже после 2-х месяцев эксплуатации.

Проектный раздел с информационным обеспечением задачи, описанием информационной модели, перечислением используемых классификаторов и системы кодирования, а также с характеристика первичных документов с нормативно-справочной и входной оперативной информацией и баз данных.

Программное обеспечение задачи, входящее в проектный раздел, включает описание логики программ, а также модулей, технологическим обеспечением задачи и руководство пользователя.

Рекомендации:

В дальнейшем возможно реализовать следующие функции в системе:

1. Расширить функционал, включив дополнительные средства аналитики, отчётности и обратной связи.
2. Провести исследование по интеграции модуля с другими корпоративными системами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Головлева, Е. Л. Корпоративные коммуникации: история и современность : учебник / Е. Л. Головлева. — Москва : МосГУ, 2021. — ISBN 978-5-907410-47-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/259334 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 157.;

2. IX Российская научно-методическая конференция. Материалы IX Российской научно-методической конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (г. Самара, 5-8 апреля 2021 г.) : материалы конференции. — Самара : ПГУТИ, 2021. — ISBN 978-5-907336-16-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/411632 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 74.;

3. Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях: материалы XXV Республиканской научной конференции студентов и аспирантов (Гомель, 21–23 марта 2022 г.) : материалы конференции / редакторы С. П. Жогаль [и др.]. — Гомель : ГГУ имени Ф. Скорины, 2022. — 323 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/329672 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 120.;

4. Журналистика, реклама и PR в современной цифровой среде: сборник материалов региональной научно-практической конференции (Архангельск, 23 апреля 2021 г.) : материалы конференции / составитель Л. В. Зайцева. — Архангельск : САФУ, 2021. — ISBN 978-5-261-01540-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/226907 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 251.;

5. Баланов, А. Н. Автоматизация, цифровизация и оптимизация бизнес-процессов: IT-решения и стратегии для современных компаний : учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — ISBN 978-5-507-48741-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/394532 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 20.;

6. Куликов, А. А. Разработка клиент-серверных приложений : учебное пособие / А. А. Куликов. — Москва : РТУ МИРЭА, 2023. — Часть 1. — ISBN 978-5-7339-1858-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/382556 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 32.;

7. Баланов, А. Н. Бэкенд-разработка веб-приложений: архитектура, проектирование и управление проектами : учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург : Лань, 2024. — ISBN 978-5-507-48818-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/394556 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 22.;

8. Вержаковская, М. А. Экономика программной инженерии. Теория, алгоритмы, программы : учебное пособие / М. А. Вержаковская, В. Ю. Аронов. — Самара : ПГУТИ, 2022. — 150 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/411530 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 69.;

9. Крюков, Д. А. Информационное обеспечение предприятия: Практикум : учебное пособие / Д. А. Крюков. — Москва : РТУ МИРЭА, 2023. — ISBN 978-5-7339-1854-9. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/382430 (дата

обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 18.;

10. (Актуальные проблемы методики обучения информатике и математике в современной школе : материалы конференции / под редакцией Л. Л. Босовой, Д. И. Павлова. — Москва : МПГУ, 2020. — ISBN 978-5-4263-0919-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/174914 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 155.).;

11. (Новые математические методы и компьютерные технологии в проектировании, производстве и научных исследованиях: материалы XXVI Республиканской научной конференции студентов и аспирантов (Гомель, 20–22 марта 2023 года) : материалы конференции : в 2 частях / редакторы С. П. Жогаль [и др.]. — Гомель : ГГУ имени Ф. Скорины, 2023 — Часть 2 — 2023. — 232 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/342386 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 91.).;

12. Кузенкова, Г. В. WEB-технологии. Разработка сайтов : учебное пособие / Г. В. Кузенкова. — Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2020. — 50 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/144688 (дата обращения: 05.06.2024). — Режим доступа: для авториз. пользователей. — С. 32.;

13. Digital 2021: Russian Federation [Электронный ресурс]. — Режим доступа: https://datareportal.com/reports/digital-2021-russian-federation (дата обращения: 05.06.2024).;

14. Digital 2022: Russian Federation [Электронный ресурс]. — Режим доступа: https://datareportal.com/reports/digital-2022-russian-federation (дата обращения: 05.06.2024).;

15. Digital 2023: Russian Federation [Электронный ресурс]. — Режим доступа: https://datareportal.com/reports/digital-2023-russian-federation (дата обращения: 05.06.2024).;

16. Digital 2024: Russian Federation [Электронный ресурс]. — Режим доступа: https://datareportal.com/reports/digital-2024-russian-federation (дата обращения: 05.06.2024).;

17. Яндекс.Вордстат [Электронный ресурс]. — Режим доступа: https://wordstat.yandex.ru/ (дата обращения: 05.06.2024).;

18. Python Telegram Bot Documentation [Электронный ресурс]. — Режим доступа: https://docs.python-telegram-bot.org/en/stable/index.html (дата обращения: 05.06.2024).;

19. whatsapp-web.js [Электронный ресурс]. — Режим доступа: https://github.com/pedroslopez/whatsapp-web.js (дата обращения: 05.06.2024).;

20. Viber Bot API (Python) [Электронный ресурс]. — Режим доступа: https://developers.viber.com/docs/api/python-bot-api/ (дата обращения: 05.06.2024).

Сценарий диалога для бота парсинга новостей.

1) Запуск бота:

Пользователь: /start

Бот: Привет! Я бот для парсинга агроновостей. Используй команды /pars для начала парсинга, /stop для остановки, /addmod для добавления модератора, /delmod для удаления модератора и /logs для просмотра логов. Только владелец и модераторы могут управлять мной.

2) Начало парсинга новостей:

Пользователь: /pars

Бот: Начинаю парсинг новостей...

Бот (лог): Parsing started by user ID: <user_id>

3) Остановка бота:

Пользователь: /stop

Бот: У вас нет прав для выполнения этой команды.

(Если команду выполняет владелец)

Бот: Бот остановлен.

Бот (лог): Bot stopped by owner

4) Добавление модератора:

Пользователь: /addmod 12345

Бот: Модератор 12345 добавлен.

Бот (лог): Moderator 12345 added by owner

(Если модератор уже существует)

Бот: Модератор 12345 уже существует.

(Если команду выполняет не владелец)

Бот: У вас нет прав для выполнения этой команды.

5) Удаление модератора:

Пользователь: /delmod 12345

Бот: Модератор 12345 удален.

Бот (лог): Moderator 12345 removed by owner

(Если модератора не существует)

Бот: Модератор 12345 не найден.

(Если команду выполняет не владелец)

Бот: У вас нет прав для выполнения этой команды.

6) Список модераторов:

Пользователь: /listmods

Бот: Список модераторов: 12345 67890

(Если команду выполняет не владелец)

Бот: У вас нет прав для выполнения этой команды.

7) Просмотр логов:

Пользователь: /logs

Бот: Отправка лога... (отправляет файл bot.log)

(Если команду выполняет не владелец)

Бот: У вас нет прав для выполнения этой команды.

Сценарий диалога для бота управления тикетами.

1) Запуск бота:

Пользователь: /start

Бот: Привет, [username]! Этот бот предназначен для обратной связи. Выберите действие ниже.

(Кнопки: "Создать тикет", "Мои тикеты")

2) Создание тикета:

Пользователь: (нажимает кнопку "Создать тикет")

Бот: Опишите вашу проблему. Ожидайте ответа от сотрудника.

Пользователь: У меня проблемы с доступом к аккаунту.

Бот: Ваш тикет #1 создан. Мы скоро свяжемся с вами.

Бот (в канал модераторов): Новый тикет #1 от пользователя 12345: У меня проблемы с доступом к аккаунту.

3) Просмотр тикетов:

Пользователь: (нажимает кнопку "Мои тикеты")

Бот: Ваши тикеты: #1: У меня проблемы с доступом к аккаунту (статус: open)

(Кнопка: "Закрыть тикет #1")

4) Закрытие тикета пользователем:

Пользователь: (нажимает кнопку "Закрыть тикет #1")

Бот: Ваш тикет #1 был закрыт.

Бот (в канал модераторов): Тикет #1 от пользователя 12345 был закрыт пользователем.

5) Взятие тикета в работу модератором:

Модератор: (нажимает кнопку "Взять в работу" в канале модераторов)

Бот (в канал модераторов): Тикет #1 от пользователя 12345 взят в работу модератором moderator_username.

Бот (модератору): Вы взяли в работу тикет #1. Вы можете общаться с пользователем здесь.

(Кнопка: "Закрыть тикет")

6) Общение модератора с пользователем:

Модератор: Пожалуйста, уточните, в чем именно проблема с доступом к аккаунту.

Бот (пользователю): Сообщение от модератора: Пожалуйста, уточните, в чем именно проблема с доступом к аккаунту.

Пользователь: Я не могу войти в аккаунт, пишет неверный пароль.

Бот (модератору): Сообщение от пользователя: Я не могу войти в аккаунт, пишет неверный пароль.

7) Закрытие тикета модератором:

Модератор: (нажимает кнопку "Закрыть тикет")

Бот (модератору): Пожалуйста, введите причину закрытия тикета или нажмите кнопку 'Закрыть тикет без причины' еще раз для закрытия без причины.

(Кнопка: "Закрыть тикет без причины")

Модератор: Проблема решена, пароль сброшен.

Бот (пользователю): Ваш тикет #1 был закрыт. Причина: Проблема решена, пароль сброшен.

Бот (модератору): Тикет #1 был закрыт с причиной: Проблема решена, пароль сброшен.

Бот (в канал модераторов): Тикет #1 от пользователя 12345 закрыт модератором moderator_username. Причина: Проблема решена, пароль сброшен.

Сценарий диалога для бота отправки новостей.

1) Запуск бота:

Пользователь: /start

Бот: Привет! Я бот для отправки и модерации новостей. Вы можете отправить новость на рассмотрение модераторам. Для обратной связи используйте [этого бота](https://t.me/efkochat_bot).

2) Начало отправки новости:

Пользователь: Отправить новость

Бот: Введите вашу новость или нажмите /cancel для отмены. Вы также можете прикрепить файл.

3) Отправка новости:

Пользователь: Сегодня был проведен важный семинар по агрономии.

Бот: Ваша новость отправлена на рассмотрение.

Бот (в канал модераторов): Сегодня был проведен важный семинар по агрономии.

(Кнопки: "Одобрить", "Отклонить", "Редактировать")

4) Редактирование новости модератором:

Модератор: (нажимает кнопку "Редактировать")

Бот (модератору): Отправьте новый текст для новости или нажмите /cancel для отмены.

Модератор: Сегодня был проведен важный семинар по агрономии с участием ведущих специалистов.

Бот (модератору): Текст новости обновлен. Вы можете одобрить или отклонить новость.

Бот (в канал модераторов): Сегодня был проведен важный семинар по агрономии с участием ведущих специалистов.

(Кнопки: "Одобрить", "Отклонить", "Редактировать")

5) Отклонение новости с указанием причины:

Модератор: (нажимает кнопку "Отклонить")

Бот (модератору): Пожалуйста, укажите причину отклонения новости.

Модератор: Новость недостаточно информативна.

Бот (модератору): Причина отклонения отправлена автору новости.

Бот (пользователю): Ваша новость была отклонена по следующей причине: Новость недостаточно информативна.

6) Одобрение новости модератором:

Модератор: (нажимает кнопку "Одобрить")

Бот (модератору): Новость опубликована.

Бот (пользователю): Ваша новость была одобрена и опубликована.

Бот (пересылает новость в канал публикаций): Сегодня был проведен важный семинар по агрономии с участием ведущих специалистов.

Код программы для бота парсинга новостей.

```
import os
import feedparser
import telebot
import sqlite3
import logging
from logging.handlers import RotatingFileHandler
import threading
from time import sleep
from bs4 import BeautifulSoup
from dotenv import load_dotenv
from datetime import datetime, timezone, timedelta
from functools import wraps
from dateutil import parser
# Загрузка переменных окружения
load_dotenv("TOKEN.env")
TELEGRAM_BOT_TOKEN = os.getenv('TELEGRAM_BOT_TOKEN')
TELEGRAM_CHAT_ID = os.getenv('TELEGRAM_CHAT_ID')
OWNER_ID = int(os.getenv('OWNER_ID'))
# Инициализация бота
bot = telebot.TeleBot(TELEGRAM_BOT_TOKEN)
# Настройка логирования с ротацией файлов
handler = RotatingFileHandler('bot.log', maxBytes=5000000, backupCount=2)
logging.basicConfig(handlers=[handler], level=logging.INFO,
format='%(asctime)s - %(message)s')
# Блокировка для работы с базой данных
db_lock = threading.Lock()
# Инициализация базы данных SQLite
def init_db():
    with db_lock:
        conn = sqlite3.connect('bot_data.db')
        cursor = conn.cursor()
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS moderators (
                id INTEGER PRIMARY KEY
            )
        ''')
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS posted_links (
                link TEXT PRIMARY KEY
            )
        ''')
        conn.commit()
        conn.close()
# Добавление модератора в базу данных
def add_moderator(mod_id):
    with db_lock:
        conn = sqlite3.connect('bot_data.db')
        cursor = conn.cursor()
        cursor.execute('INSERT INTO moderators (id) VALUES (?)', (mod_id,))
        conn.commit()
        conn.close()
# Удаление модератора из базы данных
def remove_moderator(mod_id):
    with db_lock:
        conn = sqlite3.connect('bot_data.db')
        cursor = conn.cursor()
        cursor.execute('DELETE FROM moderators WHERE id = ?', (mod_id,))
        conn.commit()
```

```
        conn.close()
# Получение списка модераторов из базы данных
def get_moderators():
    with db_lock:
        conn = sqlite3.connect('bot_data.db')
        cursor = conn.cursor()
        cursor.execute('SELECT id FROM moderators')
        mods = cursor.fetchall()
        conn.close()
    return [mod[0] for mod in mods]
# Проверка, является ли пользователь владельцем или модератором
def is_authorized(user_id):
    return user_id == OWNER_ID or user_id in get_moderators()

# Проверка, была ли ссылка уже опубликована
def is_link_posted(link):
    with db_lock:
        conn = sqlite3.connect('bot_data.db')
        cursor = conn.cursor()
        cursor.execute('SELECT 1 FROM posted_links WHERE link = ?', (link,))
        result = cursor.fetchone()
        conn.close()
    return result is not None
# Отметка ссылки как опубликованной
def mark_link_as_posted(link):
    with db_lock:
        conn = sqlite3.connect('bot_data.db')
        cursor = conn.cursor()
        cursor.execute('INSERT INTO posted_links (link) VALUES (?)', (link,))
        conn.commit()
        conn.close()
# Получение новостей из RSS-канала
def fetch_news():
    RSS_FEED_URL = 'https://xn--80aecvxfbbnpl.xn--p1ai/rss'
    feed = feedparser.parse(RSS_FEED_URL)
    news_items = []
    for entry in feed.entries:
        summary = BeautifulSoup(entry.summary, 'html.parser').get_text()
        news_items.append({
            'title': entry.title,
            'link': entry.link,
            'published': entry.published,
            'summary': summary
        })
    # Сортировка новостей по дате
    news_items.sort(key=lambda x: x['published'])
    return news_items
# Конвертация времени публикации в московское время и форматирование его
def format_time(published):
    published_time = parser.parse(published)
    moscow_time = published_time.astimezone(timezone(timedelta(hours=3)))
    return moscow_time.strftime('%Y-%m-%d %H:%M:%S')
# Отправка новости в канал Telegram
def send_news(news_item):
    if not is_link_posted(news_item['link']):
        formatted_time = format_time(news_item['published'])
        message = f"📢 {news_item['title']}\n🗓 {formatted_time} GMT+3\n🔗
{news_item['link']}\n\n{news_item['summary']}"
        bot.send_message(TELEGRAM_CHAT_ID, message)
        mark_link_as_posted(news_item['link'])
        logging.info(f"Posted news: {news_item['title']} -
{news_item['link']}")
```

```
# Отправка самой последней новости в канал Telegram
def send_latest_news():
    news_items = fetch_news()
    if news_items:
        latest_news_item = news_items[-1]
        formatted_time = format_time(latest_news_item['published'])
        message = f"📢 {latest_news_item['title']}\n🗓 {formatted_time}
GMT+3\n📎 {latest_news_item['link']}\n\n{latest_news_item['summary']}"
        bot.send_message(TELEGRAM_CHAT_ID, message)
        logging.info(f"Reposted latest news: {latest_news_item['title']} -
{latest_news_item['link']}")
# Функция для непрерывного парсинга новостей
def continuous_parsing():
    while True:
        news_items = fetch_news()
        for news_item in news_items:
            send_news(news_item)
        sleep(60)  # Ожидание 1 минуту перед следующей проверкой новостей
# Ограничение на количество запросов (антиспам)
user_requests = {}
def limit_requests(limit, interval):
    def decorator(func):
        @wraps(func)
        def wrapper(message):
            user_id = message.from_user.id
            current_time = datetime.now().timestamp()
            if user_id not in user_requests:
                user_requests[user_id] = []
            user_requests[user_id] = [timestamp for timestamp in
user_requests[user_id] if current_time - timestamp < interval]
            if len(user_requests[user_id]) < limit:
                user_requests[user_id].append(current_time)
                return func(message)
            else:
                bot.send_message(message.chat.id, "Слишком много запросов.
Пожалуйста, попробуйте позже.")
        return wrapper
    return decorator
# Команда /start
@bot.message_handler(commands=['start'])
def start(message):
    bot.send_message(message.chat.id, "Привет! Я бот для парсинга
агроновостей. Используй команды /pars для начала парсинга, /stop для
остановки, /addmod для добавления модератора, /delmod для удаления модератора
и /logs для просмотра логов. Только владелец и модераторы могут управлять
мной.")
    logging.info("Bot started")
# Команда /pars с использованием многопоточности
@bot.message_handler(commands=['pars'])
@limit_requests(limit=3, interval=60)
def pars(message):
    if is_authorized(message.from_user.id):
        bot.send_message(message.chat.id, "Начинаю парсинг новостей...")
        logging.info("Parsing started by user ID: %s", message.from_user.id)
        parsing_thread = threading.Thread(target=continuous_parsing)
        parsing_thread.daemon = True
        parsing_thread.start()
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Команда /pars222 для повторной публикации последней новости
```

```
@bot.message_handler(commands=['pars222'])
@limit_requests(limit=3, interval=60)
def pars222(message):
    if is_authorized(message.from_user.id):
        bot.send_message(message.chat.id, "Повторная публикация последней
новости...")
        logging.info("Reposting latest news by user ID: %s",
message.from_user.id)
        send_latest_news()
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Команда /stop
@bot.message_handler(commands=['stop'])
@limit_requests(limit=3, interval=60)
def stop(message):
    if message.from_user.id == OWNER_ID:
        bot.send_message(message.chat.id, "Бот остановлен.")
        logging.info("Bot stopped by owner")
        exit()
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Команда /addmod
@bot.message_handler(commands=['addmod'])
@limit_requests(limit=3, interval=60)
def addmod(message):
    if message.from_user.id == OWNER_ID:
        try:
            mod_id = int(message.text.split()[1])
            if mod_id not in get_moderators():
                add_moderator(mod_id)
                bot.send_message(message.chat.id, f"Модератор {mod_id}
добавлен.")
                logging.info(f"Moderator {mod_id} added by owner")
            else:
                bot.send_message(message.chat.id, f"Модератор {mod_id} уже
существует.")
        except (IndexError, ValueError):
            bot.send_message(message.chat.id, "Используйте команду /addmod
<ID>")
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Команда /delmod
@bot.message_handler(commands=['delmod'])
@limit_requests(limit=3, interval=60)
def delmod(message):
    if message.from_user.id == OWNER_ID:
        try:
            mod_id = int(message.text.split()[1])
            if mod_id in get_moderators():
                remove_moderator(mod_id)
                bot.send_message(message.chat.id, f"Модератор {mod_id}
удален.")
                logging.info(f"Moderator {mod_id} removed by owner")
            else:
                bot.send_message(message.chat.id, f"Модератор {mod_id} не
найден.")
        except (IndexError, ValueError):
            bot.send_message(message.chat.id, "Используйте команду /delmod
<ID>")
```

```
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Команда /listmods
@bot.message_handler(commands=['listmods'])
@limit_requests(limit=3, interval=60)
def listmods(message):
    if message.from_user.id == OWNER_ID:
        mod_list = '\n'.join(map(str, get_moderators()))
        bot.send_message(message.chat.id, f"Список модераторов:\n{mod_list}")
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Команда /logs
@bot.message_handler(commands=['logs'])
@limit_requests(limit=3, interval=60)
def logs(message):
    if message.from_user.id == (OWNER_ID):
        with open('bot.log', 'r') as log_file:
            bot.send_document(message.chat.id, log_file)
    else:
        bot.send_message(message.chat.id, "У вас нет прав для выполнения этой
команды.")
# Инициализация базы данных и запуск бота
if __name__ == "__main__":
    init_db()
    bot.polling(none_stop=True)
```

Код программы для бота управления тикетами.

```
import logging
import os
import json
import sqlite3
from dotenv import load_dotenv
from telegram import Update, InlineKeyboardMarkup, InlineKeyboardButton
from telegram.ext import Application, CommandHandler, MessageHandler,
filters, CallbackQueryHandler, ContextTypes, ConversationHandler
from telegram.error import BadRequest
from datetime import datetime
# Загрузка переменных окружения
load_dotenv('TOKEN.env')
TELEGRAM_BOT_TOKEN = os.getenv('TELEGRAM_BOT_TOKEN')
OWNER_ID = os.getenv('OWNER_ID')
MODERATOR_CHANNEL_ID = os.getenv('MODERATOR_CHANNEL_ID')
if not all([TELEGRAM_BOT_TOKEN, OWNER_ID, MODERATOR_CHANNEL_ID]):
    raise ValueError("One or more environment variables are missing.")
try:
    MODERATOR_CHANNEL_ID = int(MODERATOR_CHANNEL_ID)
except ValueError:
    raise ValueError("MODERATOR_CHANNEL_ID must be an integer.")
# Включаем логирование
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)
logger = logging.getLogger(__name__)
# Логирование загруженных переменных окружения
logger.info("TELEGRAM_BOT_TOKEN: %s", TELEGRAM_BOT_TOKEN)
logger.info("OWNER_ID: %s", OWNER_ID)
logger.info("MODERATOR_CHANNEL_ID: %s", MODERATOR_CHANNEL_ID)
# Настройка базы данных
conn = sqlite3.connect('tickets.db')
cursor = conn.cursor()
cursor.execute('''
CREATE TABLE IF NOT EXISTS tickets (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    moderator_id INTEGER,
    status TEXT NOT NULL,
    messages TEXT NOT NULL,
    created_at TEXT NOT NULL,
    updated_at TEXT NOT NULL
)
''')
conn.commit()
# Константы состояний для ConversationHandler
TYPING_REASON = range(1)
def save_ticket(user_id, moderator_id, status, messages, created_at,
updated_at):
    messages_json = json.dumps(messages)
    cursor.execute('''
        INSERT INTO tickets (user_id, moderator_id, status, messages,
created_at, updated_at)
        VALUES (?, ?, ?, ?, ?, ?)
    ''', (user_id, moderator_id, status, messages_json, created_at,
updated_at))
    conn.commit()
```

```
    return cursor.lastrowid
def get_ticket(ticket_id):
    cursor.execute('SELECT * FROM tickets WHERE id = ?', (ticket_id,))
    row = cursor.fetchone()
    if row:
        return {
            'id': row[0],
            'user_id': row[1],
            'moderator_id': row[2],
            'status': row[3],
            'messages': json.loads(row[4]),
            'created_at': row[5],
            'updated_at': row[6]
        }
    return None
def get_user_tickets(user_id):
    cursor.execute('SELECT id FROM tickets WHERE user_id = ?', (user_id,))
    rows = cursor.fetchall()
    return [row[0] for row in rows]
def update_ticket(ticket_id, updates):
    query = 'UPDATE tickets SET '
    params = []
    for key, value in updates.items():
        query += f'{key} = ?, '
        params.append(value)
    query = query.rstrip(', ') + ' WHERE id = ?'
    params.append(ticket_id)
    cursor.execute(query, params)
    conn.commit()
async def notify_moderator_channel(context: ContextTypes.DEFAULT_TYPE,
ticket_id, message, remove_buttons=False):
    try:
        if remove_buttons:
            await context.bot.edit_message_text(
                chat_id=MODERATOR_CHANNEL_ID,
                message_id=ticket_id,
                text=message
            )
        else:
            await context.bot.send_message(
                chat_id=MODERATOR_CHANNEL_ID,
                text=message,
                reply_markup=InlineKeyboardMarkup([
                    [InlineKeyboardButton("Взять в работу",
callback_data=f'handle_ticket:{ticket_id}')],
                    [InlineKeyboardButton("Закрыть тикет",
callback_data=f'close_ticket:{ticket_id}')]
                ])
            )
    except BadRequest as e:
        logger.error("Failed to send message to moderator channel: %s. Check
if the chat ID is correct and the bot has the right permissions.", e)
    except Exception as e:
        logger.error("Unexpected error: %s", e)
# Определяем команды бота
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Отправляет сообщение при вызове команды /start."""
    user = update.effective_user
    keyboard = [
        [InlineKeyboardButton("Создать тикет",
callback_data='create_ticket')],
        [InlineKeyboardButton("Мои тикеты", callback_data='view_tickets')]
```

```
    ]
    reply_markup = InlineKeyboardMarkup(keyboard
    await update.message.reply_html(
        rf'Привет, {user.mention_html()}! Этот бот предназначен для обратной
связи. Выберите действие ниже.',
        reply_markup=reply_markup
    )
async def create_ticket(update: Update, context: ContextTypes.DEFAULT_TYPE) -
> None:
    """Начинает процесс создания тикета."""
    query = update.callback_query
    await query.answer()
    await query.edit_message_text(
        text="Опишите вашу проблему. Ожидайте ответа от сотрудника."
    )
    context.user_data['creating_ticket'] = True
    logger.info("User %s is creating a ticket.", query.from_user.id)
async def receive_message(update: Update, context: ContextTypes.DEFAULT_TYPE)
-> None:
    """Принимает сообщения от пользователя и модераторов."""
    user_message = update.message.text
    user_id = update.message.from_user.id
    # Проверяем, создается ли новый тикет пользователем
    if context.user_data.get('creating_ticket', False):
        messages = [{'sender': 'user', 'text': user_message, 'timestamp':
str(datetime.now())}]
        created_at = updated_at = str(datetime.now())
        ticket_id = save_ticket(user_id, None, 'open', messages, created_at,
updated_at)
        context.user_data['creating_ticket'] = False
        await notify_moderator_channel(
            context,
            ticket_id,
            f'Новый тикет #{ticket_id} от пользователя {user_id}:
{user_message}'
        )
        await update.message.reply_text(f'Ваш тикет #{ticket_id} создан. Мы
скоро свяжемся с вами.')
    else:
        # Проверяем, модератор ли отправляет сообщение в чат тикета
        cursor.execute('SELECT id FROM tickets WHERE moderator_id = ? AND
status = ?', (user_id, 'in_progress'))
        row = cursor.fetchone()
        if row:
            ticket_id = row[0]
            ticket = get_ticket(ticket_id)
            ticket['messages'].append({'sender': 'moderator', 'text':
user_message, 'timestamp': str(datetime.now())})
            ticket['updated_at'] = str(datetime.now())
            update_ticket(ticket_id, {'messages':
json.dumps(ticket['messages']), 'updated_at': ticket['updated_at']})
            await context.bot.send_message(
                chat_id=ticket['user_id'],
                text=f'Сообщение от модератора: {user_message}'
            )
            return
        # Проверяем, пользователь ли отправляет сообщение в чат тикета
        cursor.execute('SELECT id FROM tickets WHERE user_id = ? AND status =
?', (user_id, 'in_progress'))
        row = cursor.fetchone()
        if row:
            ticket_id = row[0]
```

```
            ticket = get_ticket(ticket_id)
            ticket['messages'].append({'sender': 'user', 'text':
user_message, 'timestamp': str(datetime.now())})
            ticket['updated_at'] = str(datetime.now())
            update_ticket(ticket_id, {'messages':
json.dumps(ticket['messages']), 'updated_at': ticket['updated_at']})
            await context.bot.send_message(
                chat_id=ticket['moderator_id'],
                text=f'Сообщение от пользователя: {user_message}'
            )
            return
        await update.message.reply_text('Используйте команду /start, чтобы
начать.')
async def handle_ticket(update: Update, context: ContextTypes.DEFAULT_TYPE) -
> None:
    """Обработка тикета модератором."""
    query = update.callback_query
    await query.answer()
    logger.info("Received callback query data: %s", query.data)
    try:
        data = query.data.split(':')
        action = data[0]
        ticket_id = int(data[1])
        moderator_id = query.from_user.id
        ticket = get_ticket(ticket_id)
        if action == 'handle_ticket' and ticket and ticket['status'] ==
'open':
            update_ticket(ticket_id, {'status': 'in_progress',
'moderator_id': moderator_id, 'updated_at': str(datetime.now())})
            await context.bot.send_message(
                chat_id=moderator_id,
                text=f'Вы взяли в работу тикет #{ticket_id}. Вы можете
общаться с пользователем здесь.',
                reply_markup=InlineKeyboardMarkup([
                    [InlineKeyboardButton("Закрыть тикет",
callback_data=f'close_ticket:{ticket_id}')]
                ])
            )
            await notify_moderator_channel(
                context,
                ticket_id,
                f"Тикет #{ticket_id} от пользователя {ticket['user_id']} взят
в работу модератором {query.from_user.username}.",
                remove_buttons=True
            )
            logger.info("Ticket #%s is now in progress by moderator %s.",
ticket_id, query.from_user.username)
        else:
            logger.warning("Invalid action or ticket status for ticket #%s by
moderator %s.", ticket_id, query.from_user.username)
    except Exception as e:
        logger.error("Error handling ticket: %s", e)
async def close_ticket(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
None:
    """Инициирует закрытие тикета модератором."""
    query = update.callback_query
    await query.answer()
    logger.info("Received callback query data: %s", query.data)
    try:
        data = query.data.split(':')
        action = data[0]
        ticket_id = int(data[1])
```

```
        context.user_data['ticket_id_to_close'] = ticket_id
        # Проверяем, если тикет уже закрыт
        ticket = get_ticket(ticket_id)
        if ticket and ticket['status'] == 'closed':
            await query.edit_message_text(text="Тикет уже закрыт.")
            return ConversationHandler.END
        await query.edit_message_text(text="Пожалуйста, введите причину
закрытия тикета или нажмите кнопку 'Закрыть тикет' еще раз для закрытия без
причины.",
                                      reply_markup=InlineKeyboardMarkup([
                                          [InlineKeyboardButton("Закрыть
тикет без причины", callback_data=f'confirm_close_ticket:{ticket_id}')]
                                      ]))
        return TYPING_REASON
    except Exception as e:
        logger.error("Error initiating ticket closure: %s", e)
        return ConversationHandler.END
async def receive_closing_reason(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """Принимает причину закрытия тикета от модератора."""
    reason = update.message.text
    moderator_id = update.message.from_user.id
    ticket_id = context.user_data.get('ticket_id_to_close')
    ticket = get_ticket(ticket_id)
    if ticket and ticket['status'] != 'closed':
        ticket['status'] = 'closed'
        ticket['updated_at'] = str(datetime.now())
        ticket['messages'].append({'sender': 'moderator', 'text': f'Причина
закрытия: {reason}', 'timestamp': str(datetime.now())})
        update_ticket(ticket_id, {'status': 'closed', 'messages':
json.dumps(ticket['messages']), 'updated_at': ticket['updated_at']})
        await context.bot.send_message(
            chat_id=ticket['user_id'],
            text=f'Ваш тикет #{ticket_id} был закрыт. Причина: {reason}'
        )
        await context.bot.send_message(
            chat_id=moderator_id,
            text=f'Тикет #{ticket_id} был закрыт с причиной: {reason}'
        )
        await notify_moderator_channel(
            context,
            ticket_id,
            f"Тикет #{ticket_id} от пользователя {ticket['user_id']} закрыт
модератором {update.message.from_user.username}. Причина: {reason}",
            remove_buttons=True
        )
        logger.info("Ticket #%s closed by moderator %s with reason: %s",
ticket_id, update.message.from_user.username, reason)
    else:
        await update.message.reply_text("Ошибка: тикет не найден или уже
закрыт.")
    return ConversationHandler.END
async def confirm_close_ticket(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """Подтверждает закрытие тикета без причины."""
    query = update.callback_query
    await query.answer()
    ticket_id = int(query.data.split(':')[1])
    moderator_id = query.from_user.id
    ticket = get_ticket(ticket_id)
    if ticket and ticket['status'] != 'closed':
        ticket['status'] = 'closed'
```

```
        ticket['updated_at'] = str(datetime.now())
        ticket['messages'].append({'sender': 'moderator', 'text': 'Тикет
закрыт без указания причины', 'timestamp': str(datetime.now())})
        update_ticket(ticket_id, {'status': 'closed', 'messages':
json.dumps(ticket['messages']), 'updated_at': ticket['updated_at']})
        await context.bot.send_message(
            chat_id=ticket['user_id'],
            text=f'Ваш тикет #{ticket_id} был закрыт.'
        )
        await context.bot.send_message(
            chat_id=moderator_id,
            text=f'Тикет #{ticket_id} был закрыт без указания причины.'
        )
        await notify_moderator_channel(
            context,
            ticket_id,
            f"Тикет #{ticket_id} от пользователя {ticket['user_id']} закрыт
модератором {query.from_user.username}.",
            remove_buttons=True
        )
        logger.info("Ticket #%s closed by moderator %s without reason.",
ticket_id, query.from_user.username)
    else:
        await query.edit_message_text(text="Ошибка: тикет не найден или уже
закрыт.")
    return ConversationHandler.END

async def user_close_ticket(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    """Закрытие тикета пользователем."""
    query = update.callback_query
    await query.answer()
    ticket_id = int(query.data.split(':')[1])
    ticket = get_ticket(ticket_id)
    if ticket and ticket['status'] != 'closed':
        update_ticket(ticket_id, {'status': 'closed', 'updated_at':
str(datetime.now())})
        await context.bot.send_message(
            chat_id=ticket['user_id'],
            text=f'Ваш тикет #{ticket_id} был закрыт.')
        await notify_moderator_channel(
            context,
            ticket_id,
            f"Тикет #{ticket_id} от пользователя {ticket['user_id']} был
закрыт пользователем.",
            remove_buttons=True)
        logger.info("Ticket #%s closed by user %s.", ticket_id,
ticket['user_id'])
    else:
        await query.edit_message_text(text="Ошибка: тикет не найден или уже
закрыт.")
async def view_tickets(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
None:
    """Отображает все тикеты пользователя."""
    query = update.callback_query
    await query.answer()
    user_id = query.from_user.id
    user_tickets = get_user_tickets(user_id)
    if not user_tickets:
        await query.edit_message_text(text="У вас нет тикетов.")
        return
    keyboard = []
```

```
    message = "Ваши тикеты:\n"
    for tid in user_tickets:
        ticket = get_ticket(tid)
        message += f"#{tid}: {ticket['messages'][0]['text']} (статус:
{ticket['status']})\n"
        if ticket['status'] != 'closed':
            keyboard.append([InlineKeyboardButton(f"Закрыть тикет #{tid}",
callback_data=f'user_close_ticket:{tid}')])
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(text=message, reply_markup=reply_markup)
def main() -> None:
    """Запуск бота."""
    application = Application.builder().token(TELEGRAM_BOT_TOKEN).build()
    # Регистрируем команды
    application.add_handler(CommandHandler("start", start))
    # Регистрируем обработчики обратных вызовов
    application.add_handler(CallbackQueryHandler(create_ticket,
pattern='^create_ticket$'))
    application.add_handler(CallbackQueryHandler(handle_ticket,
pattern='^handle_ticket:[0-9]+$'))
    application.add_handler(CallbackQueryHandler(close_ticket,
pattern='^close_ticket:[0-9]+$'))
    application.add_handler(CallbackQueryHandler(confirm_close_ticket,
pattern='^confirm_close_ticket:[0-9]+$'))
    application.add_handler(CallbackQueryHandler(view_tickets,
pattern='^view_tickets$'))
    application.add_handler(CallbackQueryHandler(user_close_ticket,
pattern='^user_close_ticket:[0-9]+$'))
    # Регистрируем обработчик сообщений
    application.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
receive_message))
    # Регистрируем ConversationHandler для закрытия тикетов
    application.add_handler(ConversationHandler(
        entry_points=[CallbackQueryHandler(close_ticket,
pattern='^close_ticket:[0-9]+$')],
        states={
            TYPING_REASON: [MessageHandler(filters.TEXT & ~filters.COMMAND,
receive_closing_reason)]
        },
        fallbacks=[],
        per_chat=True,
        per_message=False
    ))
    # Запускаем бота
    application.run_polling()
if __name__ == '__main__':
    main()
```

Код программы для бота отправки новостей.

```
import os
from dotenv import load_dotenv
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup
from telegram.ext import ApplicationBuilder, CommandHandler, MessageHandler,
CallbackQueryHandler, filters, CallbackContext, ConversationHandler
import logging
# Установка конфигурации логирования
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO
)
# Загрузка переменных окружения из .env файла
load_dotenv('TOKEN.env')
# Получение переменных окружения с проверкой
try:
    BOT_TOKEN = os.getenv('BOT_TOKEN')
    CHANNEL_ID = int(os.getenv('CHANNEL_ID'))
    FORWARD_CHANNEL_ID = int(os.getenv('FORWARD_CHANNEL_ID'))
    OWNER_ID = int(os.getenv('OWNER_ID'))
except (TypeError, ValueError) as e:
    logging.error(f'Ошибка при загрузке переменных окружения: {e}')
    raise
# Константы для состояний
EDIT_NEWS, CONFIRM_EDIT, REJECT_REASON = range(3)

# Обработчик команды /start
async def start(update: Update, context: CallbackContext):
    # Актуальное описание бота и заглушка для ссылки на другого бота
    description = (
        'Привет! Я бот для отправки и модерации новостей.\n\n'
        'Вы можете отправить новость на рассмотрение модераторам. '
        'Для обратной связи используйте [этого
бота](https://t.me/efkochat_bot).'
    )
    await update.message.reply_text(
        description,
        parse_mode='Markdown'
    )
# Начало отправки новости
async def send_news_start(update: Update, context: CallbackContext):
    await update.message.reply_text('Введите вашу новость или нажмите /cancel
для отмены. Вы также можете прикрепить файл.')
    context.user_data['menu'] = 'SEND_NEWS'
    return 'SEND_NEWS'
# Обработка отправленной новости
async def send_news(update: Update, context: CallbackContext):
    if context.user_data.get('editing_message'):
        # Обработка редактирования сообщения
        await edit_news(update, context)
        return
    if context.user_data.get('rejecting_reason'):
        # Обработка причины отклонения
        await reject_reason(update, context)
        return
    news_text = update.message.caption or update.message.text
    file_id = None
    file_type = None
    if update.message.document:
```

```
        file_id = update.message.document.file_id
        file_type = 'document'
    elif update.message.photo:
        file_id = update.message.photo[-1].file_id
        file_type = 'photo'
    elif update.message.video:
        file_id = update.message.video.file_id
        file_type = 'video'
    # Сохранение данных в user_data для редактирования
    context.user_data['news_text'] = news_text
    context.user_data['file_id'] = file_id
    context.user_data['file_type'] = file_type
    context.user_data['user_id'] = update.message.from_user.id  # Сохранение ID
пользователя
    # Отправка новости в канал
    message = await send_news_to_channel(context.bot, news_text, file_id,
file_type)
    context.user_data['message_id'] = message.message_id
    context.user_data['chat_id'] = message.chat_id
    await update.message.reply_text('Ваша новость отправлена на рассмотрение.')
    context.user_data['editing_message'] = None      # Сброс состояния
редактирования
    return 'MAIN_MENU'
async def send_news_to_channel(bot, news_text, file_id, file_type):
    try:
        keyboard = [
            [
                InlineKeyboardButton("✅ Одобрить", callback_data='approve'),
                InlineKeyboardButton("❌ Отклонить", callback_data='reject')
            ],
            [InlineKeyboardButton("✏️ Редактировать", callback_data='edit')]
        ]
        reply_markup = InlineKeyboardMarkup(keyboard)
        if file_id:
            if file_type == 'photo':
                return await bot.send_photo(chat_id=CHANNEL_ID, photo=file_id,
caption=news_text, reply_markup=reply_markup)
            elif file_type == 'video':
                return await bot.send_video(chat_id=CHANNEL_ID, video=file_id,
caption=news_text, reply_markup=reply_markup)
            else:
                return       await       bot.send_document(chat_id=CHANNEL_ID,
document=file_id, caption=news_text, reply_markup=reply_markup)
        else:
            return await bot.send_message(chat_id=CHANNEL_ID, text=news_text,
reply_markup=reply_markup)
    except Exception as e:
        logging.error(f'Ошибка при отправке новости в канал: {e}')
# Обработка нажатий на кнопки модерации
async def button(update: Update, context: CallbackContext):
    query = update.callback_query
    await query.answer()
    action = query.data
    message = query.message
    user_id = context.user_data.get('user_id')  # Получение ID пользователя
    if action == 'approve':
        await forward_news(context.bot, message, 'approved')
        await query.edit_message_reply_markup(reply_markup=None)
        await query.message.reply_text("Новость опубликована.")
        if user_id:
```

```
            await context.bot.send_message(chat_id=user_id, text="Ваша новость
была одобрена и опубликована.")
    elif action == 'reject':
        await query.edit_message_reply_markup(reply_markup=None)
        await          query.message.reply_text("Новость          отклонена.",
reply_markup=InlineKeyboardMarkup([
            [InlineKeyboardButton("Указать причину", callback_data='reason')]
        ]))
    elif action == 'reason':
        await query.message.reply_text("Пожалуйста, укажите причину отклонения
новости.")
        context.user_data['reject_message'] = message
        context.user_data['rejecting_reason'] = True
        return REJECT_REASON
    elif action == 'edit':
        # Начало редактирования текста
        await query.edit_message_reply_markup(reply_markup=None)
        await query.message.reply_text("Отправьте новый текст для новости или
нажмите /cancel для отмены.")
        context.user_data['editing_message'] = message
        context.user_data['chat_id'] = message.chat_id
        context.user_data['message_id'] = message.message_id
        return EDIT_NEWS
# Обработка редактирования новости
async def edit_news(update: Update, context: CallbackContext):
    new_text = update.message.text
    chat_id = context.user_data['chat_id']
    message_id = context.user_data['message_id']
    # Обновляем текст сообщения
    keyboard = [
        [
            InlineKeyboardButton("✅ Одобрить", callback_data='approve'),
            InlineKeyboardButton("❌ Отклонить", callback_data='reject')
        ],
        [InlineKeyboardButton("✏️ Редактировать", callback_data='edit')]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    try:
        await                context.bot.edit_message_caption(chat_id=chat_id,
message_id=message_id, caption=new_text, reply_markup=reply_markup)
    except:
        await                    context.bot.edit_message_text(chat_id=chat_id,
message_id=message_id, text=new_text, reply_markup=reply_markup)

    context.user_data['editing_message'] = None
    await update.message.reply_text('Текст новости обновлен. Вы можете одобрить
или отклонить новость.')
    return ConversationHandler.END
# Обработка причины отклонения
async def reject_reason(update: Update, context: CallbackContext):
    reason = update.message.text
    message = context.user_data.get('reject_message')
    user_id = context.user_data.get('user_id')
    if user_id and message:
        await  context.bot.send_message(chat_id=user_id,  text=f"Ваша  новость
была отклонена по следующей причине: {reason}")
        await   message.reply_text("Причина   отклонения   отправлена   автору
новости.")
    context.user_data['reject_message'] = None
    context.user_data['rejecting_reason'] = False
    return ConversationHandler.END
```

```
# Отмена редактирования
async def cancel_edit(update: Update, context: CallbackContext):
    await update.message.reply_text('Редактирование отменено.')
    context.user_data['editing_message'] = None
    context.user_data['rejecting_reason'] = False
    return ConversationHandler.END
async def forward_news(bot, message, status):
    if status == 'approved':
        target_channel = FORWARD_CHANNEL_ID
        if message.photo:
            await bot.send_photo(chat_id=target_channel, photo=message.photo[-
1].file_id, caption=message.caption)
        elif message.video:
            await                        bot.send_video(chat_id=target_channel,
video=message.video.file_id, caption=message.caption)
        elif message.document:
            await                    bot.send_document(chat_id=target_channel,
document=message.document.file_id, caption=message.caption)
        else:
            await                     bot.send_message(chat_id=target_channel,
text=message.caption or message.text)
def main():
    application = ApplicationBuilder().token(BOT_TOKEN).build()
    conv_handler = ConversationHandler(
        entry_points=[
            CommandHandler("start", start),
            MessageHandler(filters.Regex("^(📰     Отправить     новость)$"),
send_news_start)
        ],
        states={
            EDIT_NEWS: [
                MessageHandler(filters.TEXT & ~filters.COMMAND, edit_news)
            ],
            REJECT_REASON: [
                MessageHandler(filters.TEXT & ~filters.COMMAND, reject_reason)
            ]
        },
        fallbacks=[CommandHandler("cancel", cancel_edit)]
    )
    application.add_handler(conv_handler)
    application.add_handler(MessageHandler(filters.TEXT  &  ~filters.COMMAND,
send_news))
    application.add_handler(MessageHandler(filters.Document.ALL             &
~filters.COMMAND, send_news))
    application.add_handler(MessageHandler(filters.PHOTO  &  ~filters.COMMAND,
send_news))
    application.add_handler(MessageHandler(filters.VIDEO  &  ~filters.COMMAND,
send_news))
    application.add_handler(CallbackQueryHandler(button))
    # Запуск бота
    application.run_polling()
if __name__ == '__main__':
    main()
```