

МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»



Направление подготовки
09.03.01 Информатика и вычислительная техника

Направленность (профиль)
«Технологии разработки программного обеспечения»

Выпускная квалификационная работа

Разработка Telegram-бота для работы линейного студенческого отряда

Обучающегося 4 курса
очной формы обучения
Леонтьевой Анна Викторовны

Руководитель выпускной квалификационной
работы:
старший преподаватель кафедры
информационных технологий и электронного
обучения
Ильина Татьяна Сергеевна

Санкт-Петербург
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
ГЛАВА 1. TELEGRAM-БОТ ДЛЯ СТУДЕНЧЕСКИХ ОТРЯДОВ.....	5
1.1. Обзор программных продуктов для работы студенческих отрядов.....	5
1.2. Инструменты и технологии для создания Telegram-бота.....	15
ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ТЕЛЕГРАМ-БОТА «ПОМОЩНИК ЛСО».....	23
2.1. Проектирование Телеграм-бота «Помощник ЛСО».....	23
2.2. Разработка Телеграм-бота «Помощник ЛСО».....	26
ЗАКЛЮЧЕНИЕ	49
СПИСОК ЛИТЕРАТУРЫ.....	50

ВВЕДЕНИЕ

В средних специальных и высших учебных заведениях для студентов и преподавателей особенно остро стоит вопрос практики и опыта работы. Одним из вариантов для студентов начать свой трудовой опыт – это работа в каникулярное время в составе Молодежной общероссийской общественной организации «Российские Студенческие Отряды». В Санкт-Петербурге ежегодно выезжают работать более 140 студенческих отрядов, общая численность которых более 4000 человек.

Одна из задач организации – воспитательная работа с участниками отрядного движения. В связи с этим, деятельность студенческих отрядов продолжается в учебное время: мероприятия, конкурсы профессионального мастерства, дополнительное обучения и многое другое. Вся перечисленная деятельность студенческих отрядов связана со сбором и обработкой информации, которой занимаются руководители студенческих отрядов – такие же студенты, основное время которых должно быть сосредоточено на получении образования. В связи с постоянным ростом количества участников отрядного движения, нагрузка по сбору и обработке информации также увеличивается. В связи с этим, было принято решение разработки Telegram-бота для организации работы, который позволит собирать и обрабатывать информацию для различных задач линейного студенческого отряда более автономно, с меньшим вмешательством руководителей студенческого отряда.

Актуальность определена заказом Штаба Студенческих отрядов РГПУ им. А. И. Герцена Telegram-бота «Помощник ЛСО». Данный проект оптимизирует процесс сбора и обработки информации.

Цель работы: разработать Telegram-бот для организации работы Линейного студенческого отряда.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ развития отрядного движения на примере Штаба Студенческих отрядов РГПУ им. А. И. Герцена;
- провести анализ существующих программных продуктов для студенческих отрядов;
- провести обзор существующих инструментов и информационных технологий для создания Telegram-бота;
- рассмотреть особенности создаваемого Telegram-бота и разработать структуру;
- разработать Telegram-бота с определенными ранее особенностями;
- разработать административную панель и базу данных для записи передаваемой пользователем информации;
- развернуть программный продукт на сервере.

Объект исследования – процесс разработки Telegram-бота.

Предметом исследования является разработка Telegram-бота «Помощник ЛСО».

Работа состоит из введения, первой главы, которая включает в себя анализ развития отрядного движения в Штаба Студенческих отрядов РГПУ им. А. И. Герцена и имеющихся информационных решений для отрядного движения, выбор технологии и инструментов разработки, и второй главы, которая описывает практическую реализацию программного продукта, заключения и списка источников.

При анализе различных источников информации предпочтение отдано научным статьям, интернет-статьям и документации программного обеспечения посвященным методам разработки программного обеспечения и особенностям разработки Telegram-ботов.

ГЛАВА 1. TELEGRAM-БОТ ДЛЯ СТУДЕНЧЕСКИХ ОТРЯДОВ

1.1. Обзор программных продуктов для работы студенческих отрядов

В России действует МООО «РСО» – Молодёжная Общероссийская общественная организация «Российские Студенческие Отряды». Целью данной организации является организация временной занятости обучающихся в общеобразовательных организациях, профессиональных образовательных организациях и образовательных организациях высшего образования, изъявивших желание в свободное от учебы время работать в различных отраслях экономики [1, пункт 2.1]. В Санкт-Петербурге действует региональное отделение МООО «РСО»: более 3500 обучающийся различных образовательных учреждений ежегодно в каникулярный период отправляются работать в составе линейных студенческих отрядов.

Студенческие отряды Санкт-Петербурга имеют определенную структуру, представленную на рисунке 1.



Рисунок 1. Структура студенческих отрядов Санкт-Петербурга

Региональный штаб представляет собой исполнительный орган власти в структуре МООО «РСО» в Санкт-Петербурге, как в субъекте Российской Федерации. Региональный штаб обеспечивает выполнение решений законодательных органов власти структуры МООО «РСО». Также региональный штаб реализует контроль за деятельностью таких структур как Штабы образовательных организаций (ШСО) и Линейные студенческие отряды.

В образовательных организациях действуют Штабы Студенческих отрядов, которые обеспечивают поддержку линейных студенческих отрядов образовательными учреждениями, а также осуществляют контроль деятельности линейных студенческих отрядов.

Линейные студенческие отряды (ЛСО) — это форма организации студентов образовательных учреждений, основная структурная единица регионального отделения МООО «РСО». Прилагательное «Линейный» обуславливается линейной структурой, где отряды разных профессиональных направлений, таких как педагогическое, строительное, археологическое, медицинское и так далее, находятся на одном уровне в структуре молодежной общественной организации (рисунок 2).

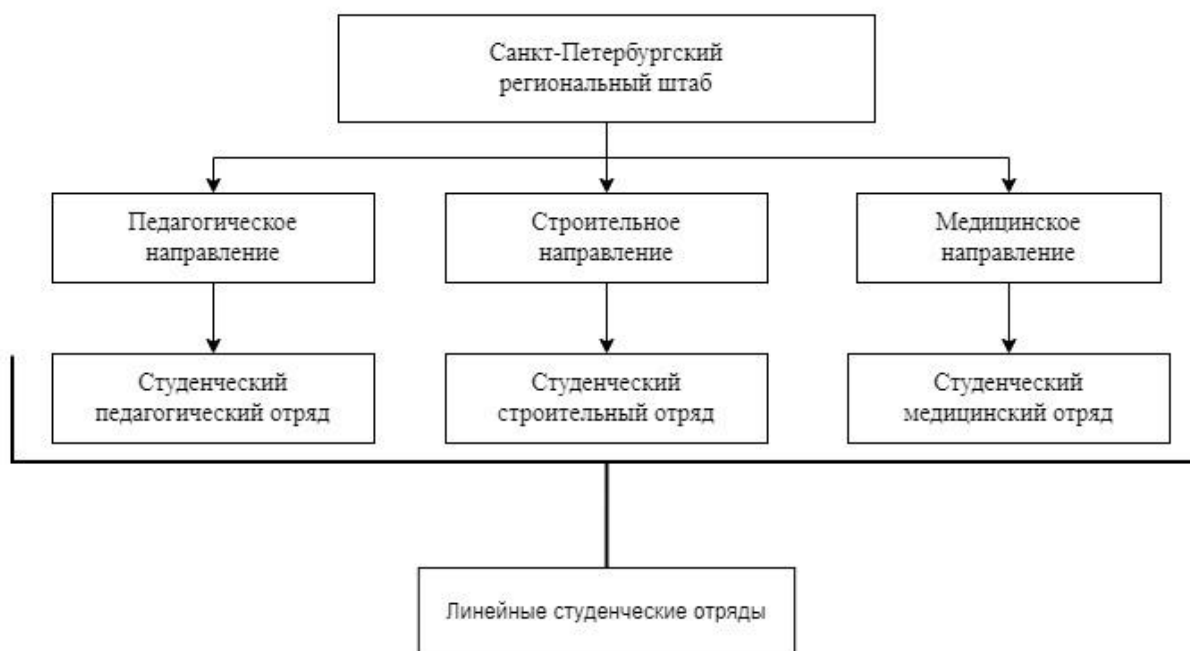


Рисунок 2. Структура направлений линейных студенческих отрядов в Санкт-Петербурге

Линейные студенческие отряды действуют круглогодично: организация мероприятий в рамках реализации воспитательной деятельности Образовательного Учреждения и МООО «РСО», организация агитационной деятельности, проведение обучения и подготовки участников движения для профессиональной деятельности в рамках трудового выезда в летнее время, сплочение коллектива отряда.

Организация деятельности в линейном студенческом отряде происходит в рамках иерархии отряда. Участниками и структурной единицей каждого отряда являются бойцы. Руководящим органом линейного студенческого отряда является командный состав, в состав которого входят:

- Командир – руководитель отряда, отвечающий за всю деятельность линейного студенческого отряда и обеспечение работы на время трудового выезда. Отвечает за сдачу отчетной информации.
- Комиссар – заместитель командира отряда, отвечающий за организацию и участие отряда в мероприятиях регионального отделения МООО «РСО» и образовательного учреждения и сплочение коллектива отряда на трудовом выезде.
- Комендант – заместитель командира отряда по хозяйственной части. Занимается обеспечением отряда атрибутикой, помещениями для учебных занятий и мероприятий, прочими материальными вопросами.
- Методист (педагогические отряды) или мастер (остальные направления отрядов) – заместитель командира по вопросу обучения и профессиональной подготовки участников отряда.
- Руководитель Пресс-центра – заместитель командира по информационной кампании отряда. Занимается организацией работы по освещению деятельности отряда и взаимодействием со сторонними средствами массовой информации.

Каждый член командного состава имеет свои обязанности, описанные в нормативно-правовых документах МООО «РСО», и, как следствие, имеет определенное количество задач по сбору и обработке информации. В таблице 1 приведены основные задачи, согласно организационно-правовому документу организации [2], членов командного состава.

Таблица 1. Основные задачи по сбору и обработке информации командного состава ЛСО

Должность в командном составе	Задача по сбору и обработке информации
Командир	Сбор информации о трудоустройстве бойцов в летний период.
	Сбор информации для оформления справок от регионального отделения МООО «РСО».
Комиссар	Сбор обратной связи по итогам мероприятий отряда.
	Сбор «Писем на сезон» в электронной версии (традиция студенческих отрядов).
Методист и Мастер	Создание методической базы отряда.
Комендант	Сбор информации по заказу атрибутики.
Руководитель пресс-центра	Сбор фото и видео материалов для публикации в интернет-ресурсах.

Задачи, связанные со сбором информации, не являются сложными задачами, требующего большого человеческого вмешательства. При этом, если численность отряда, например, более 50 человек, подобные задачи занимают большое количество времени у обучающихся, которые являются членами командного состава отряда.

Для анализа развития Молодёжной Общественной организации «Российские студенческие отряды» был взят параметр «Численность актива» и студенческие отряды Штаба Студенческих отрядов РГПУ им. А. И. Герцена. Актив отряда — это бойцы студенческого отряда, которые выехали на последний трудовой летний выезд или внесли достаточный по мнению командного состава вклад в развитие студенческого отряда. Списки актива студенческого отряда ежегодно (в сентябре) сдает командир отряда в Штаб студенческих отрядов.

Для анализа были взяты все архивные списки актива студенческих отрядов за период 2020-2023 годов. По результатам обработки данных была создана таблица 2 со сводной информацией по количеству бойцов в активе каждого студенческого отряда за каждый год. Также в таблице приведен суммарный актив Штаба студенческих отрядов РГПУ им. А. И. Герцена.

Таблица 2. Количество бойцов в активе ЛСО ШСО РГПУ им. А. И. Герцена

Студенческий отряд	Количество бойцов в активе в 2020 году	Количество бойцов в активе в 2021 году	Количество бойцов в активе в 2022 году	Количество бойцов в активе в 2023 году
Студенческий педагогический отряд "Маэстро"	52	58	58	64
Студенческий педагогический отряд "ИнКор"	58	78	91	91
Студенческий педагогический отряд "Друг"	41	44	45	51
Студенческий педагогический отряд "Кислород"	22	23	29	32
Студенческий педагогический отряд "Персик"	8	13	16	17
Студенческий строительный отряд "Рубеж"	11	10	12	16
Студенческий археологический отряд "Ворон"	25	30	35	38
Студенческий отряд проводников "Романтик"	8	13	20	23
Актив Штаба Студенческих отрядов РГПУ им. А. И. Герцена	225	269	306	332

С помощью, приведенной выше таблицы, были составлены пользовательская гистограмма «Количество бойцов в активе студенческого отряда за период 2020 - 2023 годов» (Рисунок 3) и пользовательская линейчатая диаграмма «Численность бойцов Штаба студенческих отрядов РГПУ им. А. И. Герцена за период 2020 - 2023 годов» (Рисунок 4).

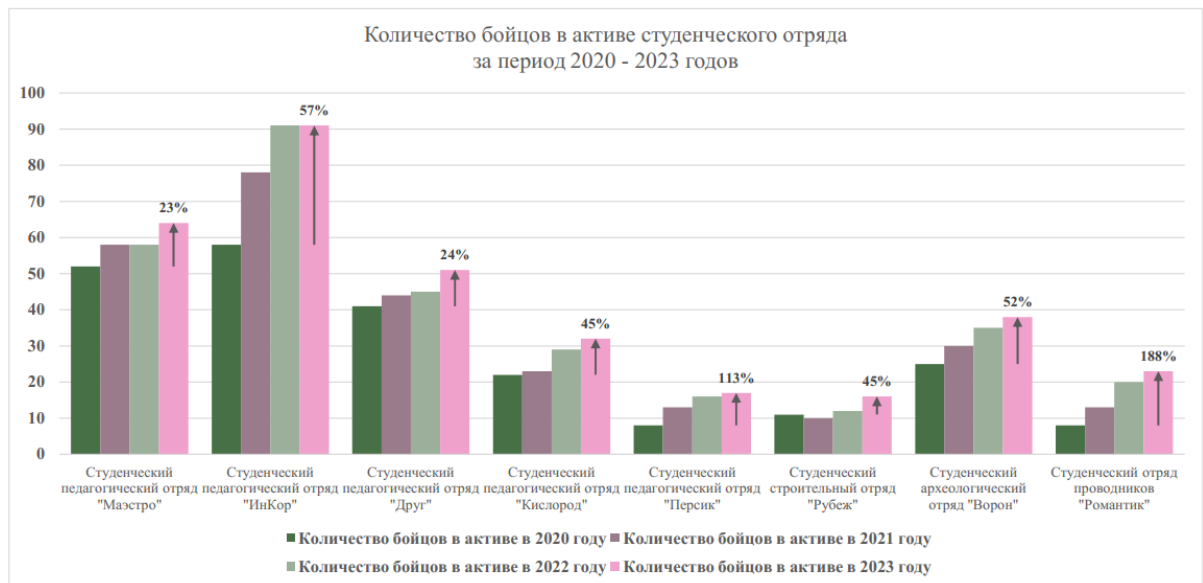


Рисунок 3. Пользовательская гистограмма «Количество бойцов в активе студенческого отряда за период 2020 - 2023 годов»

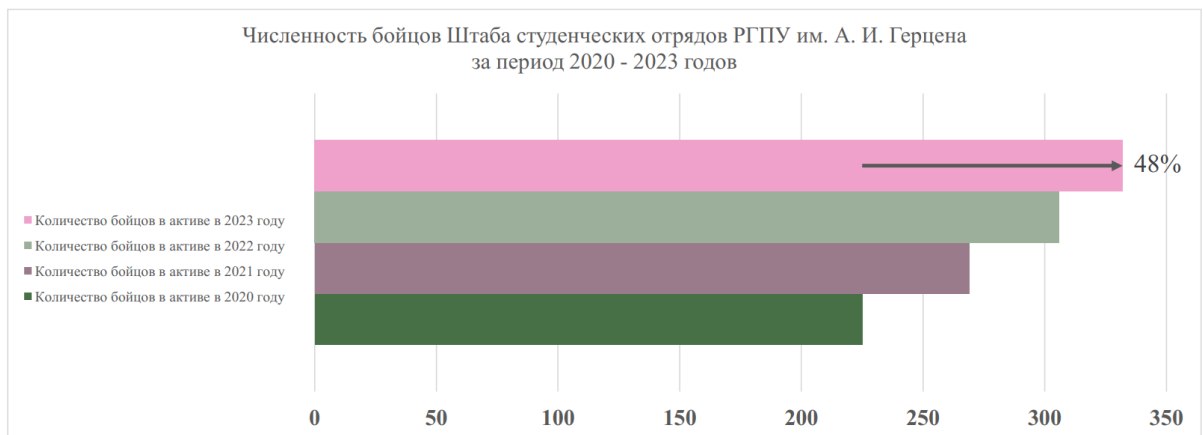


Рисунок 4. Пользовательская линейчатая диаграмма «Численность бойцов Штаба студенческих отрядов РГПУ им. А. И. Герцена за период 2020 - 2023 годов»

Диаграммы были составлены на основе информации, приведенной в таблице 2. Также были вычислен дополнительный параметр: увеличение количества бойцов в активе отрядов и штаба с 2020 года к 2023 году в процентах. Все данные отражены в диаграммах.

В ходе анализа приведенных выше диаграмм можно сделать вывод о постоянном росте количества активных бойцов в Студенческих отрядах и Штаба Студенческих отрядов РГПУ им. А. И. Герцена в целом. Исходя из факта роста количества участников движения можно сделать вывод о том, что с каждым годом нагрузка на командный состав отряда по сбору и обработке информации также увеличивается.

Рассмотрим существующие онлайн-сервисы и приложения, которые связаны с работой студенческих отрядов.

Приложение «ТрудКрут» (рисунок 5)

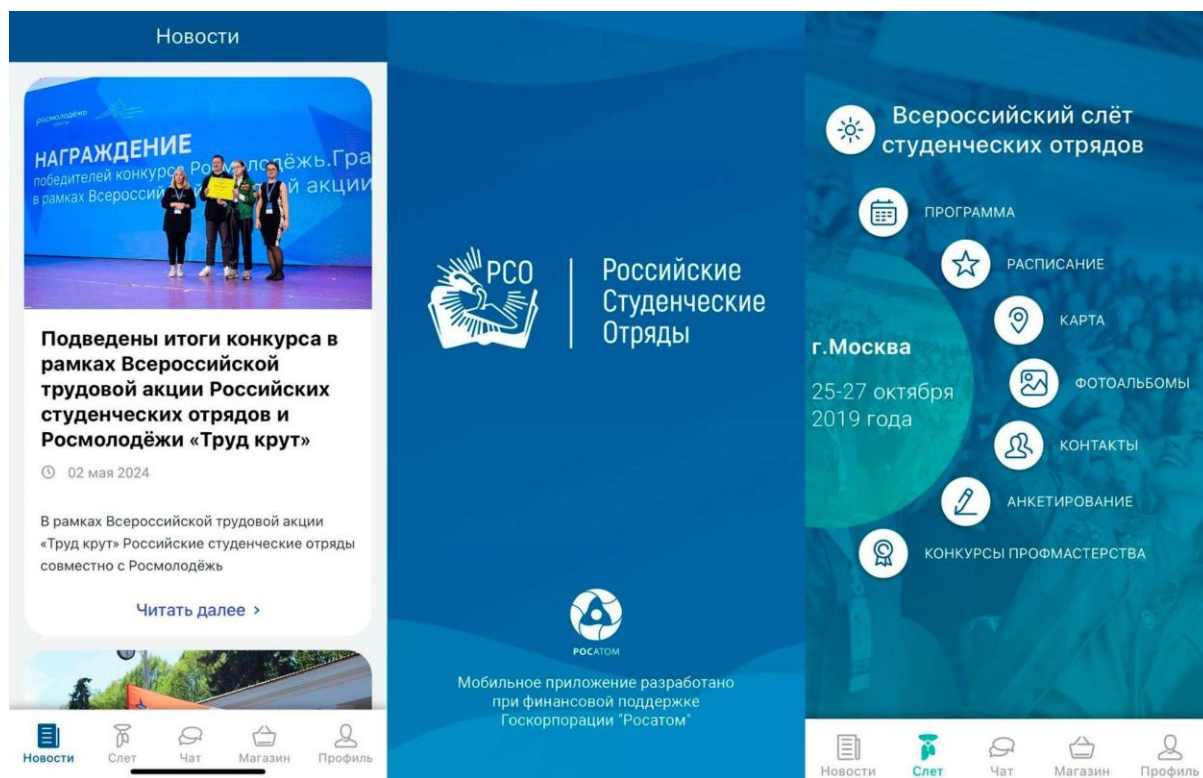


Рисунок 5. Интерфейс приложения "ТрудКрут"

Функционал приложения:

- Просмотр новостей МООО «РСО» (автоматическая публикация с сайта организации);
- Общий чат для пользователей приложения;
- Информационная поддержка Всероссийского слёта студенческих отрядов:
 - Просмотр программы всероссийского слёта;
 - Просмотр индивидуального расписания для всероссийского слёта;
 - Интерактивная карта слёта;
 - Фотоальбом слёта;
 - Контакты организаторов и кураторов слёта;
 - Анкетирование (сбор обратной связи);
 - Онлайн результаты конкурсов.

- Официальный магазин «ТрудКурт»: приложение перенаправляет пользователя на веб-сайт магазина.

Преимуществом данного приложения является удобная информационная поддержка Всероссийского слета студенческих отрядов: вся информация находится в одном месте.

Основной недостаток приложения заключается в том, что его поддержка не осуществляется с 2019 года. Также стоит отметить, что приложение не является программным продуктом, функционал которого организует работы линейного студенческого отряда. Приложение предназначено исключительно для информационной поддержки определенного мероприятия.

Приложение «Российские студенческие отряды» (рисунки 6-7)

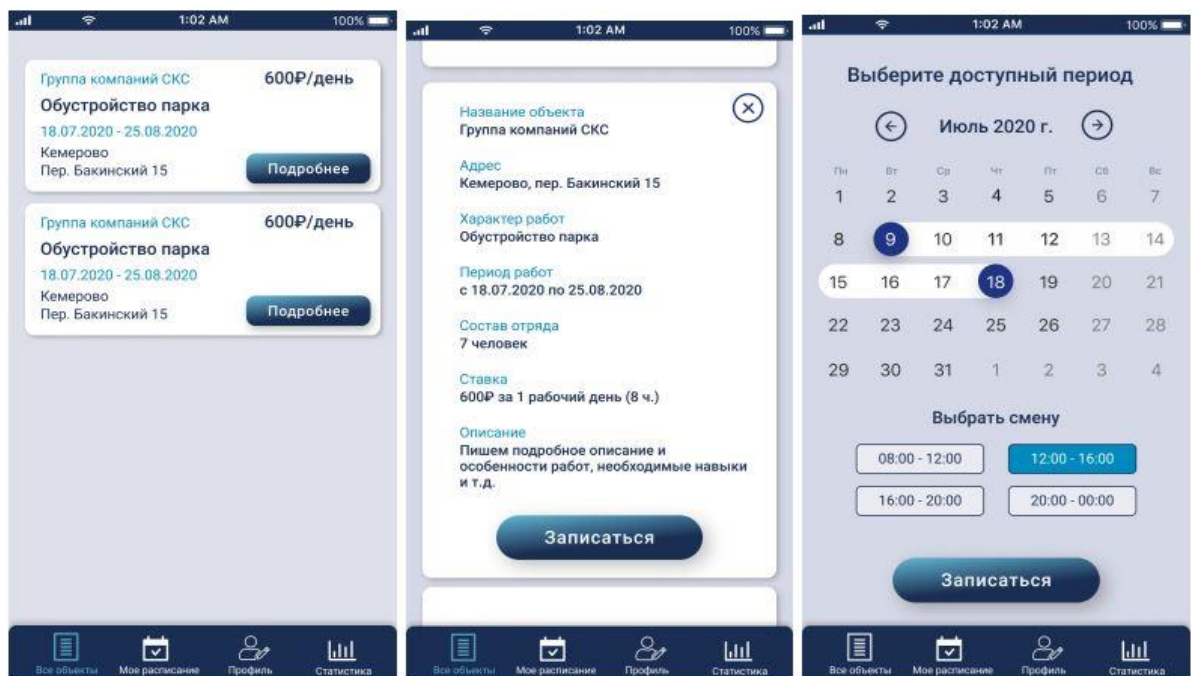


Рисунок 6. Интерфейс мобильного приложения "Российские студенческие отряды"

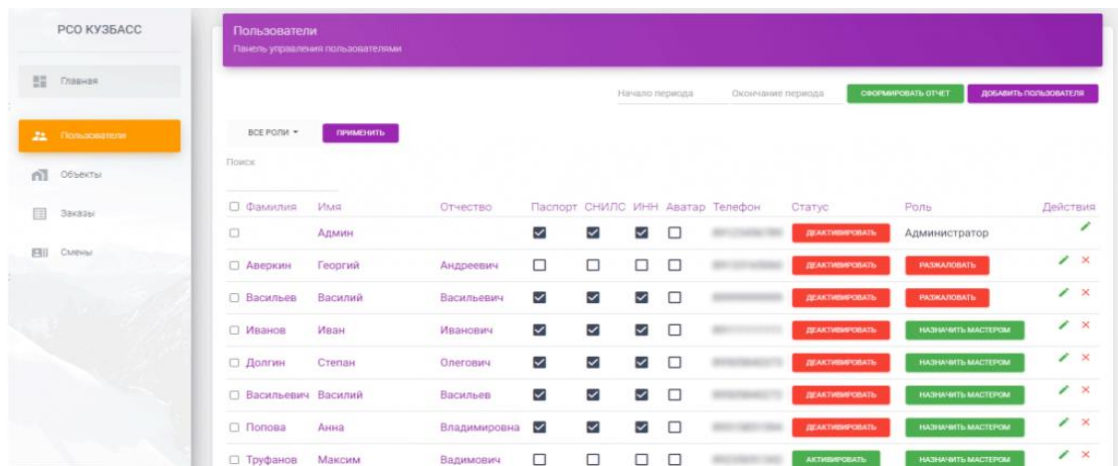


Рисунок 7. Интерфейс веб-приложения «Российские студенческие отряды»

Функционал приложения:

- База данных с данными по каждому работнику строительного объекта;
- Данные для каждого пользователя по количеству рабочих смен;
- Расчет заработной платы за летний трудовой выезд по отработанным сменам;
- Разработанная система публикации доступных рабочих смен со стороны администраторов (прораб или мастер) и возможность для пользователей на них записываться с учетом удобного времени;
- Реализованная система с базой данных отработанных смен каждого работника, которые отмечает администратор (прораб или мастер);
- Система мотивации для работников.

Приложение является удобной системой для бойцов студенческих строительных отрядов и их работодателей, особенно для масштабных всероссийских строительных проектов. Приложение не адаптировано для других направлений студенческих отрядов с похожими схемами работ: археологические, биологические, медицинские. Функционал приложения не отвечает запросу сбора и обработки информации для внутренних процессов студенческого отряда в учебный период в том числе.

VK-приложение «Я боец» (рисунок 8)

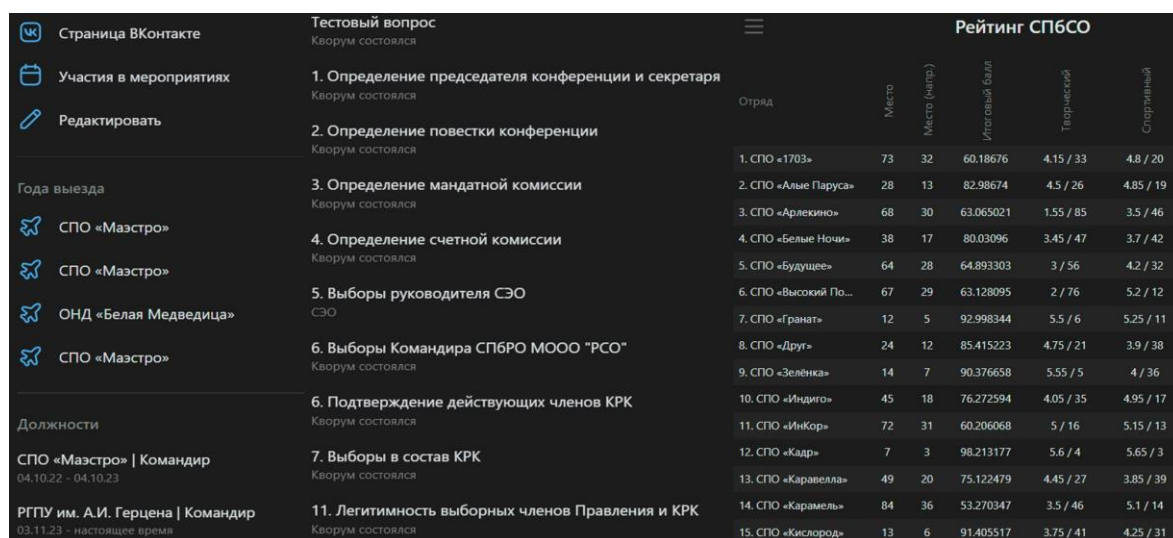


Рисунок 8. Интерфейс приложения "Я боец"

Функционал приложения:

- Календарь с предстоящими мероприятиями городского штаба и штабов образовательных учреждений (с возможностью фильтра);
- Подача заявок на городские мероприятия студенческих отрядов;
- База данных всех бойцов каждого отряда с информацией об участии в мероприятии, занимаемым должностям в командном составе и летних трудовых выездах:
 - Реализованный поиск по всем бойцам города;
 - Привязка профиля к странице социальной сети ВКонтакте;
- Реализованная площадка для проведения Конференции регионального отделения:
 - Электронное голосование, подсчет голосов, вывод результатов;
 - Контроль кворума;
- Публикация городского рейтинга Линейных студенческих отрядов.

Приложение «Я Боец» является удобным инструментом регионального штаба для реализации сбора, обработки и публикации информации для всех бойцов Санкт-Петербургского регионального отделения. Но на данный момент приложение реализует в себе функции, которые полезны исключительно в обмене информацией между командным составом линейного студенческого отряда, штабом образовательного учреждения и региональным штабом. Приложение не

предполагает сбора информации в контексте внутренних задач линейного студенческого отряда.

Автоматизация процесса сбора, обработки и хранения информации для линейных студенческих отрядов актуальна для региональных отделений с большой численностью бойцов. Санкт-Петербургское региональное отделение является один из самых больших по количеству отрядов и бойцов. В некоторых линейных студенческих отрядах Санкт-Петербурга активная часть бойцов может достигать 100 человек, что требует автоматизации процессов, связанных с работой командного состава.

1.2. Инструменты и технологии для создания Telegram-бота

На данный момент на рынке программных решений, связанных с образовательной деятельностью или деятельностью государственных общественных организаций, можно наблюдать большое количество различных вариантов продуктов: от простых веб-сайтов для конкретных задач, до полноценных мобильных приложений, иногда объединенных в целую цифровую экосистему.

Telegram-боты представляют собой программы, функционирующие в рамках мессенджера Telegram. Кроссплатформенный мессенджер Telegram, выдвинулся в лидеры рынка в последние годы благодаря общему функционалу как мессенджера, а также своим чат-ботам, предоставляющим информационные услуги.

Согласно отчёту по данным исследовательской компании Mediascope по итогам четвертого квартала 2023 Telegram остается на 4-ом месте среди российских интернет-ресурсов по объему дневной аудитории [3]. Также стоит отметить, что среди возрастной аудитории 12 лет - 24 года, Telegram занимает 1 место среди интернет-ресурсов. Так как возрастная группа студенческих отрядов заключается в возрастном периоде от 17 до 24 лет, то Telegram-бот является самым оптимальным решением для использования как платформы для программного продукта.

Выбор языка программирования и библиотеки для разработки

В настоящее время множество языков программирования находят применение в различных областях. Однако, чтобы определить оптимальный язык для создания Telegram-бота, важно ограничить выбор до нескольких наиболее распространенных языков и провести сравнительный анализ в контексте поставленной цели.

Для выбора языков программирования для проведения анализа был взят рейтинг языков программирования TIOBE [4]. Индекс популярности TIOBE формируется из подсчета поисковых запросов формата «<language> programming». Индекс TIOBE рассчитывается на основе данных из 25-ти поисковых систем. Для этого отбираются 25 веб-сайтов с высшим рейтингом по данным трекера трафика LikeWeb, которые удовлетворяют ряду критериев:

- Наличие функции поиска на домашней странице;
- Возможность отображения количества запросов на определенную страницу;
- Доступность результатов в HTML-формате с четкими тегами;
- Корректное кодирование особых символов в поисковой системе;
- Гарантия возврата как минимум 1 попадания по 1 запросу;
- Отсутствие избыточного количества аномальных результатов (выбросов) по запросу;
- Исключение веб-сайтов с противозаконным контентом.

На момент исследования первые 5 позиций занимают языки: Python, C, Java, C# и Java Script (рисунок 9):

May 2024	Programming Language		Ratings
1		Python	16.33%
2		JavaScript	9.98%
3		C++	9.53%
4		Java	8.69%
5		C#	6.49%

Рисунок 9. Топ рейтинга языков программирования ТЮВЕ, май 2024

После изучения рейтинга языков программирования ТЮВЕ, были изучены интернет-источники по данной теме: выбор языка программирования для создания Telegram-ботов. Самыми упоминаемыми языками программирования для данной задачи из приведённого выше рейтинга были Python, Java, и JavaScript [5].

JavaScript — это кроссплатформенный объектно-ориентированный язык сценариев, используемый для создания интерактивных веб-страниц. Использование JavaScript дает возможность создавать Telegram-ботов, которые могут выполнять различные задачи, такие как отправка и получение сообщений, запись и возвращение данных из базы данных, обработка команд и многое другое. Для создания чат-ботов в Telegram с использованием Java Script существует несколько инструментов и библиотек. Например, библиотека Telegraf, фреймворк Botpress, библиотека Node-telegram-bot-api.

Java является объектно-ориентированным языком программирования. Этот язык используется для разработки различных мобильных и веб-приложений, а также десктопных программ.

В Java также существует множество библиотек и фреймворков для создания ботов для Телеграм. Одними из самых популярных инструментов для создания ботов для Телеграм-ботов на Java является библиотека Telegram Bots и фреймворк Spring. Оба этих инструмента позволяют взаимодействовать с API Телеграма и имеют

инструменты, позволяющие работать с разными типами сообщений, включая текстовые сообщения, файлы разных форматов, а также множество других функций Телеграма.

Python — это высокоуровневый язык программирования общего назначения, который также используется для разработки мобильных приложений и веб-приложений. Язык нацелен на повышение производительности труда разработчиков и читабельности кода.

Для создания ботов для Телеграм на Python существует несколько библиотек, которые предоставляют различные функции и возможности. Наиболее популярными из них являются: `python-telegram-bot`, `telebot`, `aiogram`.

Python обеспечивает широкие возможности для разработки ботов для Телеграм. Он позволяет управлять взаимодействием с пользователем, обрабатывать сообщения, отправлять медиафайлы и многое другое.

В ходе анализа преимуществ языков программирования для написания Telegram-ботов, был выбран язык Python версии 3.10.11. По сравнению с остальными языками программирования, Python предлагает большее количество различных библиотек и других инструментов для написания Телеграм-бота, отличается простой структурой кода и интегрируемостью с другими технологиями разработки программного обеспечения. В качестве среды разработки выбран для использования PyCharm.

Язык Python имеет обширное количество библиотек для создания Телеграм-ботов. Для выбора библиотеки для разработки был проведен сравнительный анализ существующих и наиболее популярных в интернет-ресурсах библиотек по 3 критериям: производительность, функциональность и поддержка документации [6,7]. Были определены самые упоминаемые и популярные библиотеки в интернет-ресурсах по теме разработки Телеграм-ботов: `python-telegram-bot`, `aiogram`, `Telebot`.

- Критерий 1 «Производительность»:

- python-telegram-bot обладает высокой производительностью и поддерживает асинхронность;
 - aiogram также обладает высокой производительностью обработки запросов и поддерживает асинхронность;
 - Telebot по сравнению с перечисленными ранее библиотеками обладает средней производительностью, а также менее эффективно поддерживает асинхронность, что сказывается на обработке запросов.
- Критерий 2 «Функциональность»:
 - python-telegram-bot обладает достаточным набором инструментов для обработки различных форматов файлов и типов сообщений, а также легко интегрируется с внешними сервисами;
 - aiogram имеет большое количество инструментов для обработки различных сообщений, форматов файлов, а также событий. Легко интегрируется с внешними ресурсами;
 - Telebot подходит для разработки ботов, не требующих сложной функциональности, также для интеграции с внешними ресурсами может потребоваться более глубокая настройка, по сравнению с другими упомянутыми библиотеками.
- Критерий 3 «Поддержка документации»:
 - Документация python-telegram-bot обновляется, а также имеет активную поддержку сообщества и множество учебных материалов;
 - aiogram также имеет активное сообщество пользователей, актуальную документацию и большое количество учебных материалов и примеров кода;
 - Telebot имеет менее активное сообщество пользователей, что сказывается на количестве учебных материалов и ресурсов.

По итогам сравнительного анализа была выбрана библиотека aiogram версии 3.4.1, так как библиотека имеет большой функционал, поддерживает асинхронность, отличается простым синтаксисом и имеет большую поддержку сообщества пользователей.

Выбор технологии для создания панели администратора

Для создания панели администратора будет использована такая технология как веб-фреймворк (англ. web-Framework). В переводе фреймворк означает некоторую структуру, в рамках которой разрабатывается программный продукт. Фреймворк в веб-программировании является специальной платформой, которая делает процесс веб-разработки легче несмотря на то, что фреймворки используются и в других областях программирования. Существует множество различных фреймворков с использованием различных языков: PHP, Ruby, Java, Python и другие [8]. Самые популярные по использованию фреймворки имеют целые сообщества, учебные материалы и учебники.

Для такой задачи, как создание административной панели фреймворк подходит оптимальнее всего, так как фреймворк позволяет создавать динамические сайты, не требуя больших ресурсов, как системы управления сайтом.

Для выбора фреймворка для использования, важно отметить, что язык программирования должен совпадать с языком библиотеки, который был выбран для реализации Telegram-бота, так как это облегчит разработку, исключая написание промежуточных прослоек между двумя языками. Также при написании работы был важен факт наличия встроенного административного интерфейса и учебных материалов по созданию административной панели для чат-бота с помощью выбранного фреймворка.

По описанным выше запросам был выбран фреймворк Django. Этот фреймворк является общедоступным проектом, что является преимуществом в случае поставленной задачи: создания административной панели для относительно небольшого Телеграм-бота. На этапе начала процесса информатизации, это преимущество является одним из ключевых. Django — один из самых популярных фреймворков, поэтому существует достаточное для начала работы число сообществ помощи разработчикам, примеров и учебных материалов, в том числе – по созданию административной панели для чат-ботов [9]. В случае

развитие программного проекта Django способен справиться с высоконагруженными сайтами благодаря встроенным инструментам. Фреймворк Django позволяет разрабатывать продукт без посторонних компонентов, при этом внутри проекта существует независимость составляющих частей, которые можно заменить или модифицировать без затрагивания других компонентов.

Таким образом использование фреймворка Django является оптимальным для решения поставленной задачи.

Выбор технологии для реализации базы данных

Для реализации базы данных в контексте Django используется Object-relational mapping (ORM) – это способ доступа к реляционной базе данных с помощью объектно-ориентированного языка (например Python). ORM является прослойкой между базой данных и объектно-ориентированным кодом, реализованным разработчиком, которая позволяет созданные объекты добавлять в базу данных или извлекать уже существующие [10].

Django ORM относится к наиболее распространенному и простому типу ORM, реализующему шаблон проектирования Active Record [11]. Основной принцип заключается в том, что таблицы базы данных обернуты в классы так, что объектный экземпляр привязан к единственной строке в таблице. После создания нового объекта соответствующая ему новая строка таблицы добавляется на сохранение. Любой загруженный объект получает всю необходимую информацию из базы данных. Если объект был изменен, то строка, соответствующая ему в таблице базы данных, будет также изменена. Класс обёртки реализует методы средства доступа или свойства для каждого столбца в таблице. Использование подобного подхода делает программный продукт безопаснее, так как получается избегать прямых SQL-запросов, а также делает код продукта и его написание легче [12].

Согласно документации Django, ORM Django поддерживает следующие базы данных: PostgreSQL, MariaDB (является ответвление реляционной СУБД MySQL), MySQL, Oracle, SQLite.

Из всех вышеперечисленных баз данных было решено использовать SQLite для периода разработки и PostgreSQL для работы непосредственно на сервере. PostgreSQL имеет 2 основных в контексте работы преимущества: поддержка опытного сообщества разработчиков и расширяемость, на случай развития проекта. Базой данных для кэширования, сохранения состояний пользователя был выбран Redis, который является сервером баз данных типа ключ-значение.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ТЕЛЕГРАМ-БОТА «ПОМОЩНИК ЛСО»

2.1. Проектирование Телеграм-бота «Помощник ЛСО»

Основная задача Телеграм-бота — это автоматизирование сбора информации по основным задачам каждой должности в командном составе линейного студенческого отряда. Исходя из этого был проведен анализ возможности автоматизации с помощью Телеграм-бота тех задач, которые были описаны в пункте 1.1.1 «Описание структуры МООО «РСО».

Функции, которые были определены для реализации в Телеграм-боте:

- Сбор информации для оформления справок от регионального отделения МООО «РСО»: пользователь вводит необходимые для создания документа-справки данные, справка автоматически формируется и отправляется в чат администратору бота для дальнейшего подписания. При заполнении должна производиться проверка правильности введенных данных;
- Сбор обратной связи по итогам мероприятий отряда: пользователь выбирает мероприятие из предложенных и заполняет анкету. Результат с ответами пользователя по мероприятию сохраняется в базе данных и отображается в панели администратора;
- Сбор «Писем на сезон» в электронной версии: пользователь пишет тестовое сообщение с возможностью прикрепить изображение и вводит ник пользователя бота. Введенное письмо анонимно отправляется необходимому пользователю в чат с Телеграм-ботом;
- Сохранение методической разработки пользователем: пользователь отправляет ссылку на облачное хранилище\ другой источник, где хранится методическая разработка, заполняет анкету для классификации методической разработки. Вся информация от пользователя сохраняется в базе данных с отображением в панели администратора;

- Поиск методической разработки в базе: пользователь уточняет необходимые параметры методической разработки, по результатам заполнения Телеграм-бот отправляет список подходящих разработок из базы данных;
- Сбор информации по заказу атрибутики: пользователю предлагаются те позиции атрибутики, которые можно заказать. Пользователь выбирает нужную позицию, вводит количество и размер, данные сохраняются в базу данных и отображаются в панели администратора;
- Сбор фото и видео материалов для публикации в интернет-ресурсах: пользователь может загрузить фото и видеоматериалы на Яндекс.Диск пакетно через диалог с Телеграм-бот, создавая необходимую папку;
- Ссылка на приложение «Я боец»: добавить ссылку на приложение с аннотацией.

Диаграмма архитектуры разрабатываемого Телеграм-бота [14] приведена на рисунке 10.

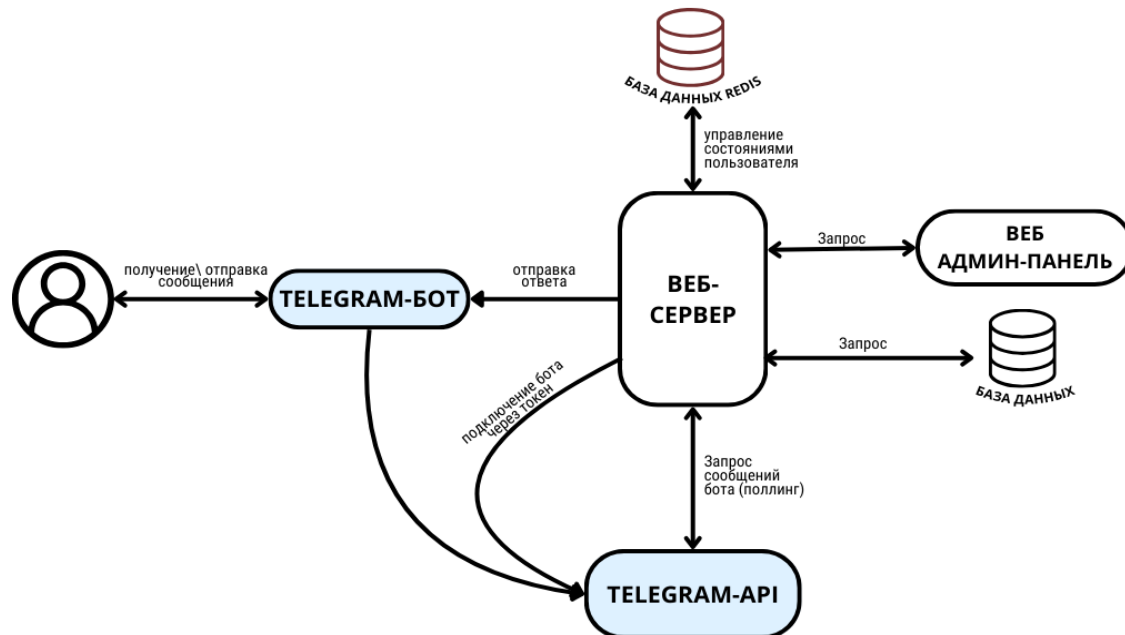


Рисунок 10. Аархитектура разрабатываемого Телеграм-бота

Пользователь отправляет команду или сообщение Телеграм-боту. При этом пользователь взаимодействует с серверами посредством обычного HTTPS-интерфейса, который является API Telegram. Сервер проверяет наличие

обновление у API Телеграма и, находя таковое, начинает обработку. Сервер может отправить запрос на запись или вывод информации у базы данных (в т.ч. Redis), а также отображает данные на панели администратора. После обработки запроса сервер отправляет ответ бота, который отображает ответ на экране пользователя. Определенные функции будут доступны пользователи в виде меню: списка кнопок, после нажатия которых будет работать определённый хендлер функции. После окончания использования функции, пользователь будет возвращаться к меню выбора.

В разработке Телеграм-ботов (а также в разработке программных продуктов, предполагающих асинхронность) используются следующие механизмы:

- Апдейт (англ. Update) – какое-либо событие, например: сообщение, редактирование сообщения, колбэк, инлайн-запрос, платёж, добавление бота в группу и другое.
- Хендлер (англ. Handler) – механизм, которые представляет собой асинхронную функцию, которая получает апдейт и обрабатывает его.
- Диспетчер (англ. Dispatcher) – объект, получающий апдейты от Telegram и определяющий, какой хендлер должен этот апдейт обработать.
- Роутер (англ. router) – объект, аналогичных диспетчеру, отвечающий за подмножество хендлеров. Таким образом диспетчер является корневым роутером (рисунок 11) [15].
- Мидлвар (англ. middleware) – промежуточное программное обеспечение, в библиотеке aiogram представляет собой механизм настройки хендлеров.
- Состояние – механизм, позволяющий запоминать текущий контекст диалога с пользователем.
- Клавиатура – элемент интерфейса, который позволяет пользователю взаимодействовать с ботом, выбирая predetermined варианты ответа.

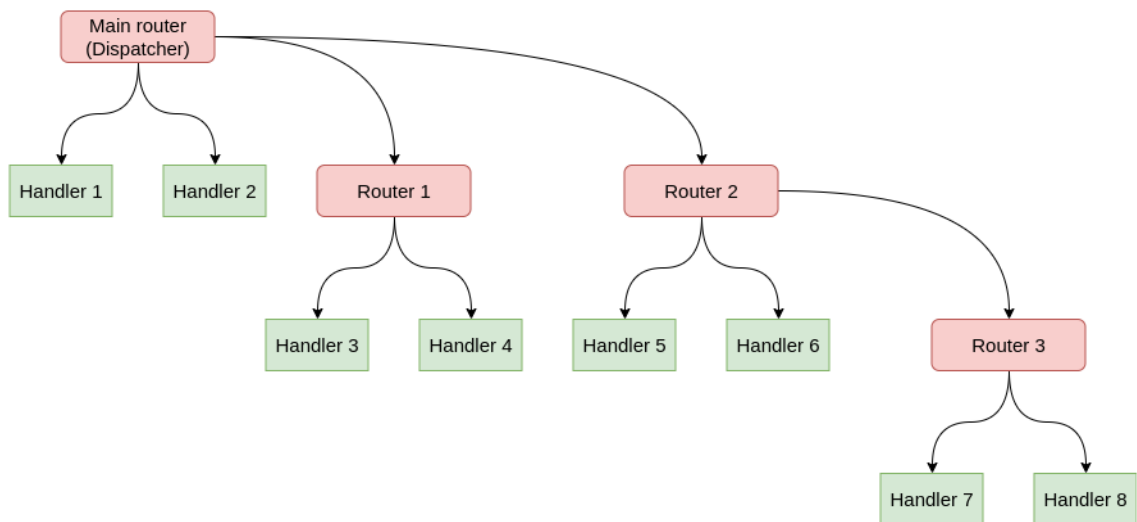


Рисунок 11. Схема взаимодействия диспетчера, роутеров и хендлеров Телеграм-бота

2.2. Разработка Телеграм-бота «Помощник ЛСО»

Для начала разработки Телеграм-бота необходимо зарегистрировать его в Телеграме в Чат-боте самого Телеграма «Bot Father». С помощью данного бота разработчик не только регистрирует свой Телеграм-бот и получает токен, но также имеет возможность редактировать описание, команды, главное изображение и многое другое. На рисунке 12 показано создание бота и получение токена:

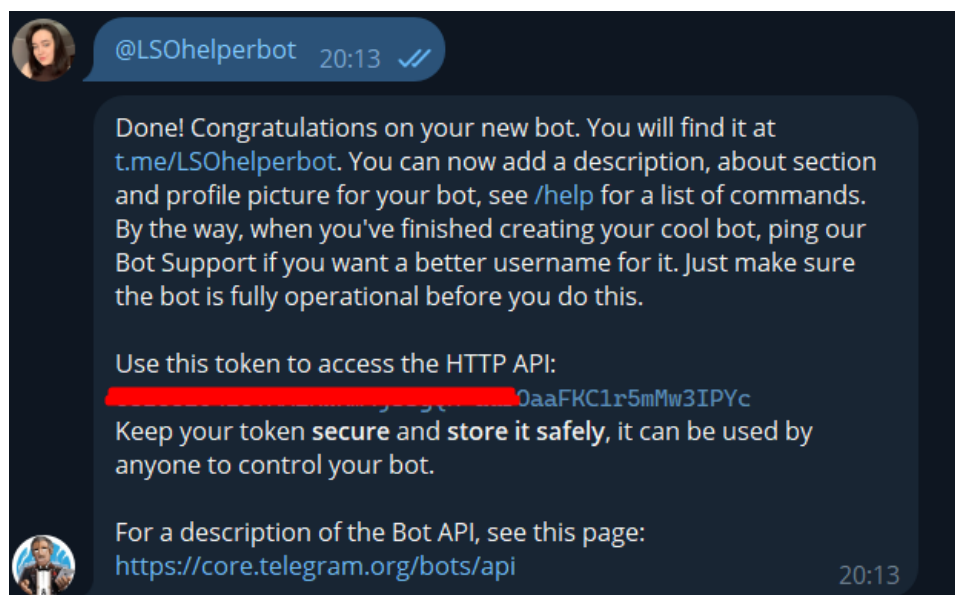
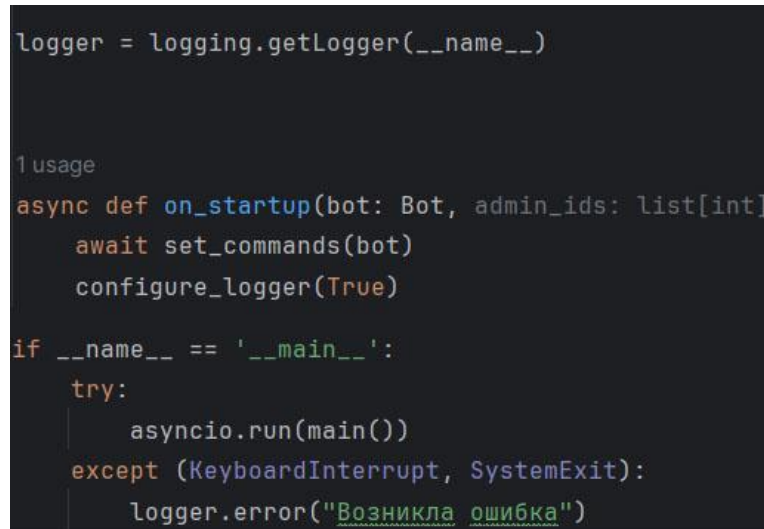


Рисунок 12. Создание Телеграм-бота и получение токена с помощью чат-бота «Bot Father»

После создания Телеграм-бот представлял собой пустой чат, куда пользователь может прислать сообщение, но ответ не будет дан. Разработка была начата с описание основного файла bot.py, в котором было необходимо описать все зависимости и запустить бот.

В файле bot.py были импортированы необходимые библиотеки, настроен запуск бота, логирование, установка команд бота (рисунок 13).

A screenshot of a code editor showing Python code for a Telegram bot. The code is as follows:

```
logger = logging.getLogger(__name__)

1 usage
async def on_startup(bot: Bot, admin_ids: list[int])
    await set_commands(bot)
    configure_logger(True)

if __name__ == '__main__':
    try:
        asyncio.run(main())
    except (KeyboardInterrupt, SystemExit):
        logger.error("Возникла ошибка")
```

Рисунок 13. Программный код с запуском Телеграм-бота

Чтобы бот начал взаимодействовать с API Telegram, необходимо указать полученный ранее токен. Все настройки конфигурации и другие параметры Телеграм-бота были записаны в файл config.py. В этом файле указаны параметры баз данных, самого Телеграм-бота. Данные для конфигурации, в том числе токен, были указаны в файле .env. Загрузка конфигурации показана на рисунке 14.

```
def load_config(path: str = None):
    env = Env()
    env.read_env(path)

    return Config(
        tg_bot=TgBot(
            token=env.str("BOT_TOKEN"),
            admin_ids=list(map(int, env.list("ADMINS"))),
            use_redis=env.bool("USE_REDIS"),
        ),
        db=DbConfig(
            host=env.str('DB_HOST'),
            password=env.str('PG_PASSWORD'),
            user=env.str('DB_USER'),
            database=env.str('DB_NAME')
        ),
        misc=Miscellaneous(
            user_redis=env.bool('USE_REDIS'),
            scheduler=AsyncIOScheduler(timezone=env.str('TIME_ZONE')),
            super_user_name=env.str('SUPER_USER_NAME'),
            super_user_pass=env.str('SUPER_USER_PASS')
        ),
        redis=Redis(
            host=env.str('REDIS_HOST'),
            port=env.int('REDIS_PORT'),
            db_fsm=env.str('REDIS_DB_FSM'),
            job_store=env.str('REDIS_DB_JOBSTORE')
        ),
    )
)
```

Рисунок 14. Загрузка конфигурации в файле bot.py

Конфигурация необходима не только при запуске бота, но и в целом для работы бота: например, для доступа к базе данных из хендлеров. Исходя из этого, был зарегистрирован мидлвар, который добавляет конфигурацию в запросы (рисунок 15). В файле Bot.py была зарегистрирована эта прослойка.

```
class ConfigMiddleware(BaseMiddleware):
    def __init__(self, config) -> None:
        self.config = config

    async def __call__(
        self,
        handler: Callable[[Message, Dict[str, Any]], Awaitable[Any]],
        event: Message,
        data: Dict[str, Any]
    ) -> Any:
        data['config'] = self.config
        return await handler(event, data)
```

РЕГИСТРАЦИЯ В BOT.PY

```
def register_global_middleware(dp: Dispatcher, config):
    dp.message.outer_middleware(ConfigMiddleware(config))
    dp.callback_query.outer_middleware(ConfigMiddleware(config))
```

Рисунок 15. Программная прослойка для добавления данных конфигурации

Для настройки Django для панели администратора в bot.py была описана функция установки Django, в которой фреймворк был импортирован, установлены

переменные окружения, разрешение асинхронные операций и инициализирован Django (рисунок 16).

```
def setup_django():
    import django

    os.environ.setdefault(
        key="DJANGO_SETTINGS_MODULE",
        value='admin_panel.admin_panel.settings'
    )
    os.environ.update({"DJANGO_ALLOW_ASYNC_UNSAFE": "true"})
    django.setup()
```

Рисунок 16. Функция установки Django

Далее в основной асинхронной функции main происходит запуск Django, запись сообщения о начале работы бота в лог, загрузка конфигурации, настройка хранилища состояний (redis), создание бота и диспетчера, регистрация глобального мидлвар, импорт и регистрация хендлеров и их роутеров, создание суперпользователя и запуск Телеграм-бота и начало поллинга на наличие обновлений от API Телеграм.

Для начала работы с ботом будет установлена команда /start, в ответ на которую бот выводит меню со списком функций рисунок(17).

```
async def set_commands(bot: Bot):
    commands = [
        BotCommand(
            command="start",
            description="Запустить бота",
        ),
    ]
    await bot.set_my_commands(commands=commands, scope=BotCommandScopeDefault())
```

Рисунок 17. Программный код установки команды /start

Для определения администрации бота используется перечисление id в Телеграм предполагаемых ботов в файле .env. Далее при работе кода бота будет работать фильтр, по которому будет проверяться, совпадает ли id использующего бота пользователя с id администратора (рисунок 18).

```
class AdminFilter(BaseFilter):
    is_admin: bool = True

    async def __call__(self, obj: Message, config: Config) -> bool:
        return (obj.from_user.id in config.tg_bot.admin_ids) == self.is_admin
```

Рисунок 18. Фильтр проверки администратора

Разработка хендлера и клавиатуры каждой функции

В Телеграм-ботах существует два основных вида кнопок [13]:

- ReplyKeyboard кнопки — это кнопки-шаблоны сообщений. При нажатии подобной кнопки в чат с ботом отправляется сообщение и, соответственно, обработка запроса происходит исходя из сравнения отправленного пользователем шаблонного сообщения. Эти кнопки отображаются внизу экрана, рядом с окном набора сообщения.
- InlineKeyboard кнопки — колбэк-кнопки, позволяющие реализовывать более сложные механизмы обработки запросов. У InlineKeyboard есть специальное значение data, по которому назначается колбэк и запрос обрабатывается. Кнопки привязываются к сообщению, отправленному ботом.

Несмотря на возможность использовать оба вида клавиатур, в ходе разработки Телеграм-бота было принято решение использовать InlineKeyboard клавиатуры, так как они не ограничены в реализуемом функционале, а также не перегружают диалог с ботом. На рисунке 19 представлен пример создания клавиатуры для основного меню.

```
async def menu_kb():
    keyboard = InlineKeyboardBuilder()
    keyboard.button(text="Заказ справки РСО", callback_data="doc_rso")
    keyboard.button(text="Обратная связь по мероприятиям", callback_data="feedbacks")
    keyboard.button(text="Письма на сезон", callback_data="seasons_letters")
    keyboard.button(text="Загрузка методических разработок", callback_data="methods")
    keyboard.button(text="Поиск методических разработок из базы", callback_data="search")
    keyboard.button(text="Заказ атрибутики", callback_data="buy")
    keyboard.button(text="Загрузка фото и видео материалов", callback_data="load_media")
    keyboard.button(text="Ссылка на приложение ЯБоец", callback_data="app_lso")
    return keyboard.adjust(1).as_markup()
```

Рисунок 19. Программный код с созданием клавиатуры основного меню

Для хранения состояний пользователя были созданы классы состояний, наследуемые от StatesGroup (aiogram), для различных хендлеров. В процессе работы хендлера пользователь «переходит» от состояния к состоянию, чтобы не терялся контекст, например при частичном перезапуске бота. Для хранения информации о переходах между состояниями используется объект FSMContext (англ. Finite State Machine Context), который позволяет управлять переходами между состояниями и хранить данные, связанные с диалогом [16]. В случае процесса обучения или разработки для работы FSMContext и записи состояний используется класс MemoryStorage. В контексте работы, для хранения данных о работе FSMContext используется Redis. В конце каждой асинхронной функции хендлера пользователю последовательно присваиваются состояния, которые обозначают, на каком шаге хендлера он находится (рисунок 20).

```
@router_letter.message(LetterStates.state_2, F.photo)
async def take_photo(message: Message, state: FSMContext):
    photo_id = message.photo[-1].file_id
    await state.update_data(photo_id=photo_id)
    await message.answer(text='Введите ник пользователя, которому хотите отправить письмо')
    await state.set_state(LetterStates.state_3)
```

Рисунок 20. Пример изменения состояния

Пример создания подобного класса состояний показан на рисунке 21.

```
class States(StatesGroup):
    state_1 = State()
    state_2 = State()
    state_3 = State()
    state_4 = State()
    state_5 = State()
    state_6 = State()
```

Рисунок 21. Пример создания класса состояний

Так как в ходе разработки Телеграм-боты было разработано 8 функций, было принято решение сначала описать в данном тексте основную суть разработки хендлеров, а далее упомянуть основные особенности каждого хендлера.

Для начала разработки хендлера были импортированные необходимые библиотеки [15]: в основном это будет библиотека aiogram, с основными для любого хендлера классами (рисунок 22):

- класс Router позволяет организовать маршрутизацию хендлеров;
- класс F является так называемым «магическим фильтром», то есть фильтром, облегчающим синтаксис создания фильтра (например, сравнения);
- класс Bot позволяет взаимодействовать с API Телеграма.
- класс FSMContext, представляющий контекст машины состояний, упомянутого ранее;
- класс CallbackQuery представляет callback-запрос от inline-кнопки.
- класс Message представляет сообщение Telegram. Содержит информацию о тексте сообщения, отправителе, времени отправки и другом.

```
from aiogram import Router, F, Bot
from aiogram.fsm.context import FSMContext
from aiogram.types import CallbackQuery, Message
```

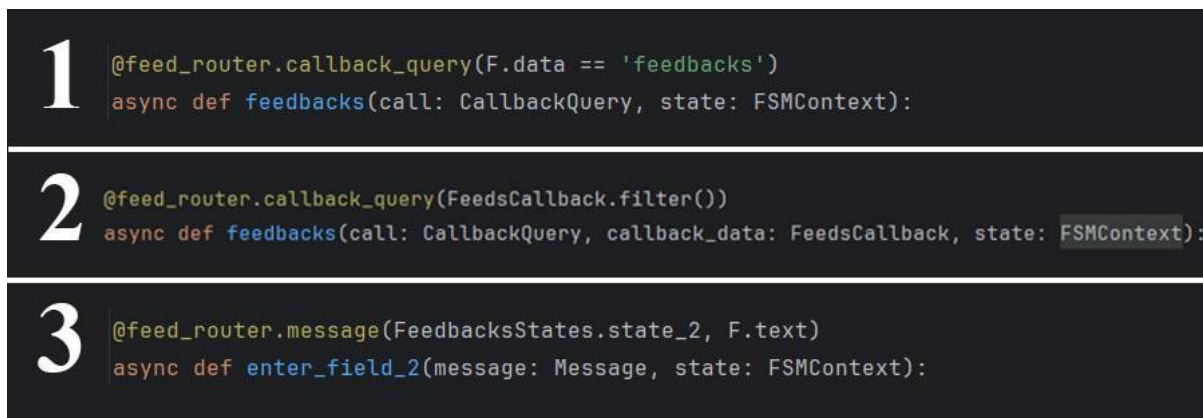
Рисунок 22. Базовый импорт хендлера

Также были импортированы модели базы данных и класс состояний. Далее был создан экземпляр класса Router() для организации маршрутизации хендлеров: чтобы хендлер был увиден и запросы на него перенаправлялись.

После импорта необходимых классов, клавиатур, связанных с хендлером, других необходимых в контексте определенного хендлера фильтров, классов и другого и регистрации роутера, происходит основной этап разработки хендера: написание функций для обработки запроса.

Каждая асинхронная функция начинается с регистрации для роутера декоратором, где прописывается, в каком случае будет срабатывать функция. На рисунке 23 под цифрой 1 представлена ситуация, когда функция обрабатывает запрос в случае нажатия inline-кнопки, data которой равна 'feedbacks'. При этом, последующие функции в этом же хендлере будут реагировать:

- В случае наличия других inline-клавиатур, на наличие их callback-запросов (рисунок 23, цифра 2);
- На изменение состояния пользователя (рисунок 23, цифра 3).



```

1 @feed_router.callback_query(F.data == 'feedbacks')
  async def feedbacks(call: CallbackQuery, state: FSMContext):

2 @feed_router.callback_query(FeedsCallback.filter())
  async def feedbacks(call: CallbackQuery, callback_data: FeedsCallback, state: FSMContext):

3 @feed_router.message(FeedbacksStates.state_2, F.text)
  async def enter_field_2(message: Message, state: FSMContext):
  
```

Рисунок 23. Примеры обработки запросов при объявлении функции хендлера

Как можно увидеть на рисунке (сверху), в случае разработки Телеграм-бота с использованием aiogram декораторы упрощают регистрацию хендлеров, а именно их функций, через роутеры.

Далее описаны особенности реализации каждой из функций.

В ходе выполнения функции «Заказ справки РСО» бот запрашивает у пользователя заранее определенную необходимую информацию для справки. Каждый вопрос анкеты представлен внутри асинхронной функции хендлера, между вопросами происходит переход между состояниями. В ходе опроса происходит проверка правильности ввода (морфологический разбор, проверка падежа) с помощью библиотеки rymorphy2. Информация пользователя записывается в переменных, которая в конце работы хендлера будет подставлена в определенные в файле some_func.py блоки (рисунок 24 и рисунок 25) файла docx. Сам файл формата docx существует внутри проекта, что позволяет использовать функцию без необходимости обращаться к другому программному обеспечению, которое поддерживает файлы формата docx [13]. В ходе опроса пользователь может выбрать шаблон справки: для мероприятия (участник\ организатор), для трудового сезона\ смены. Необходимость выбора обосновывается существованием трёх видов справок: для трудоустройства, для организатора мероприятия, для участника мероприятия;

```

doc = Document(path_str)
temp_file = tempfile.NamedTemporaryFile(delete=False, suffix='.docx')
for table in doc.tables:
    for row in table.rows:
        for cell in row.cells:
            for paragraph in cell.paragraphs:
                if 'Ректоры' in paragraph.text:
                    paragraph.text = paragraph.text.replace('Ректоры', to_job_title)
                    paragraph.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
                if 'РГПУ им. А. И. Герцена' in paragraph.text:
                    paragraph.text = paragraph.text.replace('РГПУ им. А. И. Герцена', name_university)
                    paragraph.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER
                for run in paragraph.runs:
                    run.bold = True
                if 'Тарасовы С. В.' in paragraph.text:
                    paragraph.text = paragraph.text.replace('Тарасовы С. В.', to_name)
                    paragraph.alignment = WD_PARAGRAPH_ALIGNMENT.CENTER

for paragraph in doc.paragraphs:
    if '02.02.2024' in paragraph.text:
        paragraph.text = paragraph.text.replace('02.02.2024', date)
    if date_start:
        if '-----' in paragraph.text:
            paragraph.text = paragraph.text.replace('-----', f' {date_start} ')
    if date_end:
        if '-----' in paragraph.text:
            paragraph.text = paragraph.text.replace('-----', f' {date_end}')
    if 'Иванов Иван Иванович' in paragraph.text:

```

Рисунок 24. Блоки для подстановки данных

```

name_file = await docs_editor(
    date=date,
    date_start=data.get('date_start') if data.get('date_start') else '',
    date_end=data.get('date_end') if data.get('date_end') else '',
    to_job_title=data.get('job_title'),
    name_university=data.get('name_university'),
    to_name=data.get('fullname_client'),
    user_fullname=data.get('fullname_from_client'),
    group=data.get('group'),
    name_event=event.name_event,
    fulldate=data.get('fulldate'),
    path_str=path_str
)

```

Рисунок 25. Подстановка данных файл формата docx

После сбора всей необходимой информации, данные вставляются в шаблон справок РСО формата docx с помощью модуля python-docx и готовый файл отсылается администратору бота на дальнейшую обработку документа (подпись региональным отделением).

При работе хендлера функции «Обратная связь по мероприятиям» происходит опрос пользователя по заранее введенным вопросам. Выбор мероприятия происходит с помощью базы данных с помощью асинхронного метода

BaseService.select_all_filter [15]. Так как у поля active_status есть булево значение, то выводятся для пользователя мероприятия со значением active_status = True. Ответ пользователя на каждый вопрос сохраняется в контексте машины состояний под порядковым ключом, по которому, в конце работы хендлера, ответ будет записан в базу данных с помощью метода BaseService.create_object_db() (рисунок 26).

```
@feed_router.message(FeedbacksStates.state_1, F.text)
async def enter_field_1(message: Message, state: FSMContext):
    await state.update_data(field_1=message.text)
    await message.answer(text='Что НЕ понравилось в организации бытов
                             reply_markup=await cancel_enter_feed_kb())
    await state.set_state(FeedbacksStates.state_2)

@feed_router.message(FeedbacksStates.state_6, F.text)
async def enter_field_6(message: Message, state: FSMContext):
    data = await state.get_data()
    event: EventModel = await BaseService.select_one(
        EventModel,
        pk=data.get("event_pk")
    )
    await BaseService.create_object_db(
        FeedbackModel,
        event=event,
        field_1=data.get('field_1'),
        field_2=data.get('field_2'),
        field_3=data.get('field_3'),
        field_4=data.get('field_4'),
        field_5=data.get('field_5'),
        field_6=message.text,
    )
)
```

Рисунок 26. Запись данных пользователя в машину состояний и подстановка в файл
Результат работы функции «Заказ справки РСО» представлен на рисунке 27.

Ректору 22:38 ✓

Сокращенное (по уставу) название образовательного учреждения 22:38

Отмена

РГПУ им. А. И. Герцена 22:38 ✓

Фамилия и инициалы человека в датальном падеже 22:38

Отмена

Тарасову С. В. 22:38 ✓

ФИО пользователя 22:38

Отмена

Леонтьева Анна Викторовна 22:38 ✓

Название группы 22:38

Отмена

406_ИВТ-1 22:38 ✓

Выбор: мероприятие или сезон 22:38

Меро-организатор 35.1 KB 22:39

Российский Студенческий Отряд

Санкт-Петербургское региональное отделение
Милитарской общероссийской общественной организации
19103, г. Санкт-Петербург, Сивкопольский наб., д.30 лит. 3, пом. 9Н
электронная почта: srb@moscow.ru; сайт: srb.ru
ИНН 7842/90464 КПП 7842/01001 ОГРН 1127800000096

Ректору
РГПУ им. А. И. Герцена
Тарасову С. В.

от 20.05.2024 № _____
на № _____ от _____

Справка

Данная справка свидетельствует о том, что студент Леонтьева Анна Викторовна, группа 406_ИВТ-1, 30 мая 2024 года участвует в организации мероприятия «Фестиваль ШСО РГПУ им. А. И. Герцена».

Руководитель (Командир) А.И. Чупрова

Рисунок 27. Результат работы функции «Заказ справки РСО»

В функции «Письма на сезон» пользователь набирает сообщение, по желанию отправляет изображение и вводит имя пользователя, которому необходимо отправить письмо. При вводе сообщения, отправке изображения происходит опрос пользователя с записью передаваемых данных в переменные. При проверке существования получателя письма, происходит поиск никнейма получателя в базе данных. При наличии получателя в базе, объект типа Client, представляющий искомого пользователя, сохраняется в переменную find_client. Далее из этого объекта берется идентификатор чата, куда отсылается введенной письмо (рисунок 28).

```
@router_letter.message(LetterStates.state_3, F.text)
async def check_nickname(message: Message, state: FSMContext, bot: Bot):
    nickname = message.text
    find_client: Client = await BaseService.select_one(
        Client, username=nickname
    )
    if "@" in message.text:
        return await message.answer(text="Введи ник без знака '@', пожалуйста",
                                    reply_markup=await cancel_add_kb())
    if not find_client:
        return await message.answer(text="Кажется боец с таким никнеймом не пользуется ботом: ( Проверь правильность
                                         никнейма или попроси этого бойца начать пользоваться ботом!",
                                    reply_markup=await cancel_add_kb())
    data = await state.get_data()
    await message.answer(text=f"Письмо отправлено бойцу под ником: {nickname}")
    if data.get('photo_id'):
        await state.clear()
        return await bot.send_photo(chat_id=find_client.telegram_id, photo=data.get('photo_id'),
                                    caption='Привет, боец! Тебе тут пришло письмо:\n' + data.get('text'))
    await bot.send_message(chat_id=find_client.telegram_id,
                           text='Привет боец! Тебе тут пришло письмо:\n' + data.get('text'))
    await state.clear()
```

Рисунок 28. Функция поиска получателя письма

Результат работы функции «Письма на сезон» представлен на рисунке 29.

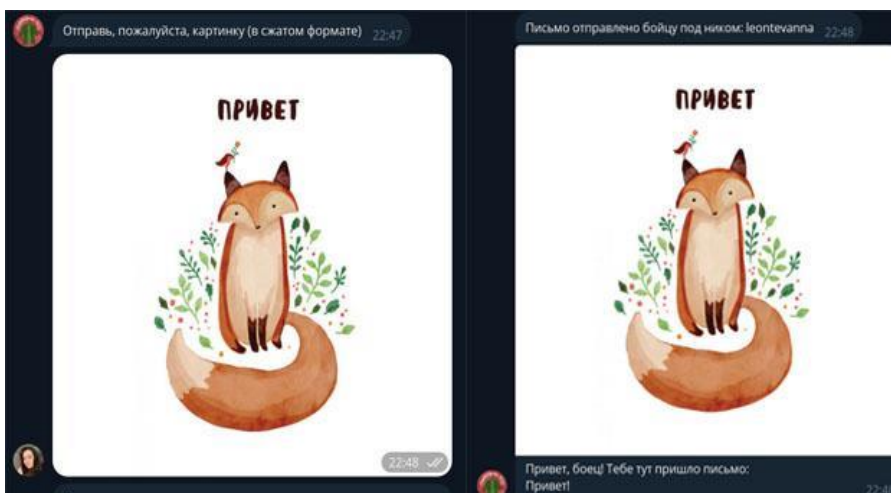


Рисунок 29. Результат работы функции «Письма на сезон»

Функция «Загрузка методических разработок» реализована таким образом, что пользователь вводит ссылку методической разработки и ее название, а далее происходит опрос по классификации разработки с помощью inline-клавиатур (рисунок 30).

```

async def age_kb():
    keyboard = InlineKeyboardBuilder()
    keyboard.button(text="Младший", callback_data="younger")
    keyboard.button(text="Средне-младший", callback_data="younger_middle")
    keyboard.button(text="Средне-старший", callback_data="middle_high")
    keyboard.button(text="Старший", callback_data="high")
    keyboard.button(text="Разновозрастный", callback_data="another")
    return keyboard.adjust(1).as_markup()

2 usages
async def age_search_kb():
    keyboard = InlineKeyboardBuilder()
    keyboard.button(text="Младший", callback_data="younger_s")
    keyboard.button(text="Средне-младший", callback_data="younger_middle_s")
    keyboard.button(text="Средне-старший", callback_data="middle_high_s")
    keyboard.button(text="Старший", callback_data="high_s")
    keyboard.button(text="Разновозрастный", callback_data="another_s")
    keyboard.button(text="Назад", callback_data="main_menu")
    return keyboard.adjust(1).as_markup()

```

Рисунок 30. Создание inline-клавиатур с классификацией методических разработок

После введения всей необходимой информации, происходит запись всех переданных пользователем данных из FSMContext в базу данных (рисунок 31).

```

age_dct = {
    'younger': "Младший",
    "younger_middle": "Средне-младший",
    "middle_high": "Средне-старший",
    "high": "Старший",
    "another": "Разновозрастный",
}

events_dct = {
    'event_1': 'Общелагерное мероприятие',
    "event_2": 'Отрядное мероприятие',
    "event_3": 'Межотрядное мероприятие',
}

development_dct = {
    'sport': 'Спортивное мероприятие',
    "creative": 'Творческое мероприятие',
    "candle": 'Свечка',
    'intellectual': 'Интеллектуальное мероприятие',
    'free': 'Свободный тип',
}

await BaseService.create_object_db(
    MethodsModel,
    url=data.get('url'),
    name=data.get('name'),
    age_str=age_dct.get(data.get('age')),
    type_event=events_dct.get(data.get('event')),
    type_develop=development_dct.get(call.data)
)

```

Рисунок 31. Запись данных из FSMContext в базу данных

При поиске разработок с помощью функции «Поиск методических разработок» пользователь также проходит опрос бота, реализованный с помощью inline-кнопок. После того, как пользователь введёт необходимые параметры по классификации разработки, происходит поиск мероприятий в базе данных, соответствующих критериям, полученным из данных data и call.data: data используется для доступа к информации, собранной на предыдущих этапах взаимодействия с пользователем, а call.data используется для получения информации о параметре, выбранном пользователем непосредственно в момент нажатия на кнопку [15]. Затем формируется список подходящих разработок и отправляется пользователю. На рисунке 32 представлен код с поиском разработок и составлением списка подходящих мероприятий.

```

events: MethodsModel = await BaseService.select_all_filter(
    MethodsModel,
    url=data.get('url'),
    name=data.get('name'),
    age_str=age_dct.get(data.get('age')),
    type_event=events_dct.get(data.get('event')),
    type_develop=development_dct.get(call.data)
)
text = '\n'.join([f'{mero.name}: {mero.url}' for mero in events])
if not text:
    return await call.message.edit_text(text="Таких разработок нет:", reply_markup=await cancel_add_kb())

```

Рисунок 32. Программный код с поиском разработок и составлением списка подходящих мероприятий

Результат работы функций «Загрузка методических разработок» и «Поиск методических разработок» представлен на рисунке 33.

Выбери мероприятие, по которому хочешь оставить отзыв: 22:43

Фестиваль ШСО РГПУ им. А. И. Герцена

Методический выезд #1

Экзамен ЛСО: 2024 год

Назад

Отзыв к мероприятию	Быт: понравилось	Быт: не понравилось	Быт: добавить	Прог: понравилось	Прог: не понравилось	Прог: добавить
Фестиваль ШСО РГПУ им. А. И. Герцена	Всё понравилось!	Такого нет!	горячий чай!	Всё понравилось!	Такого нет!	Дисотеку!

Рисунок 33. Результат работы функций «Загрузка методических разработок» и «Поиск методических разработок»

В начале работы хендлера функции «Заказ атрибутики» происходит отправка возможных для заказа предметов атрибутики. Коллаж фото, а также набор инлайн-кнопок формируется из информации из базы данных, полученной с помощью асинхронного метода `BaseService.select_all_filter` из таблицы `CollageModel` (рисунок 34).

```
@router_buy.callback_query(F.data == 'buy')
async def buy_handler(call: CallbackQuery):
    items: list[CollageModel] = await BaseService.select_all(
        CollageModel
    )
    if not items:
        return await call.answer(text="Сейчас нет товаров", show_alert=True)
    await call.answer()
    media = [
        InputMediaPhoto(media=item.photo_id) for item in items
    ]
    await call.message.answer_media_group(media)
    await call.message.answer(text='Выбери товар', reply_markup=await medias_kb(items))
```

Рисунок 34. Составление коллажа фото и инлайн-клавиатуры

Далее пользователь вводит необходимое количество предмета атрибутики и выбирает размер из предложенных вариантов inline-клавиатуры. На рисунке 35

отмечены моменты проверки количества и получения списка размеров из базы данных.

```
@router_buy.message(BuyStates.state_1, F.text)
async def count_user(message: Message, state: FSMContext):
    data = await state.get_data()
    await message.answer(text=f'Доступно для заказа {data.get("count")}')
    if message.text.isalpha():
        return await message.answer(text='Нужно ввести количество', reply_markup=await cancel_buy_kb())
    if int(data.get('count')) < int(message.text):
        return await message.answer(text='Нельзя заказать больше чем есть', reply_markup=await cancel_buy_kb())
    await state.update_data(count=message.text)
    if data.get("size"):
        sizes: list = data.get("size").split(',')
        return await message.answer(text='Выбери размер', reply_markup=await sizes_kb(sizes))
    await message.answer(text='Введи свое ФИО', reply_markup=await cancel_buy_kb())
    await state.set_state(BuyStates.state_2)
```

Рисунок 35. Проверка наличия атрибутики в базе данных

Далее, при записи заказанной атрибутики происходит получение объекта товара из базы данных по его первичному ключу (pk) и уменьшение количества товара item на количество, заказанное пользователем. Далее происходит создание нового объекта заказа (OrderModel) в базе данных с информацией о заказе (количество, размер, имя заказчика). Описанные процессы указаны на рисунке ().

```
@router_buy.message(BuyStates.state_2, F.text)
async def fullname_user(message: Message, state: FSMContext):
    data = await state.get_data()
    item: CollageModel = await BaseService.select_one(CollageModel, pk=data.get("item_pk"))
    item.quantity -= int(data.get('count'))
    item.save()

    await BaseService.create_object_db(
        OrderModel,
        quantity=int(data.get('count')),
        size=data.get('size') if data.get('size') else None,
        full_name=message.text
    )
    await message.answer(text="Ваш заказ принят")
    await user_start(message, state)
```

Рисунок 36. Регистрация заказа в базе данных

Результат работы функции «Заказ атрибутики» представлен на рисунке 37.

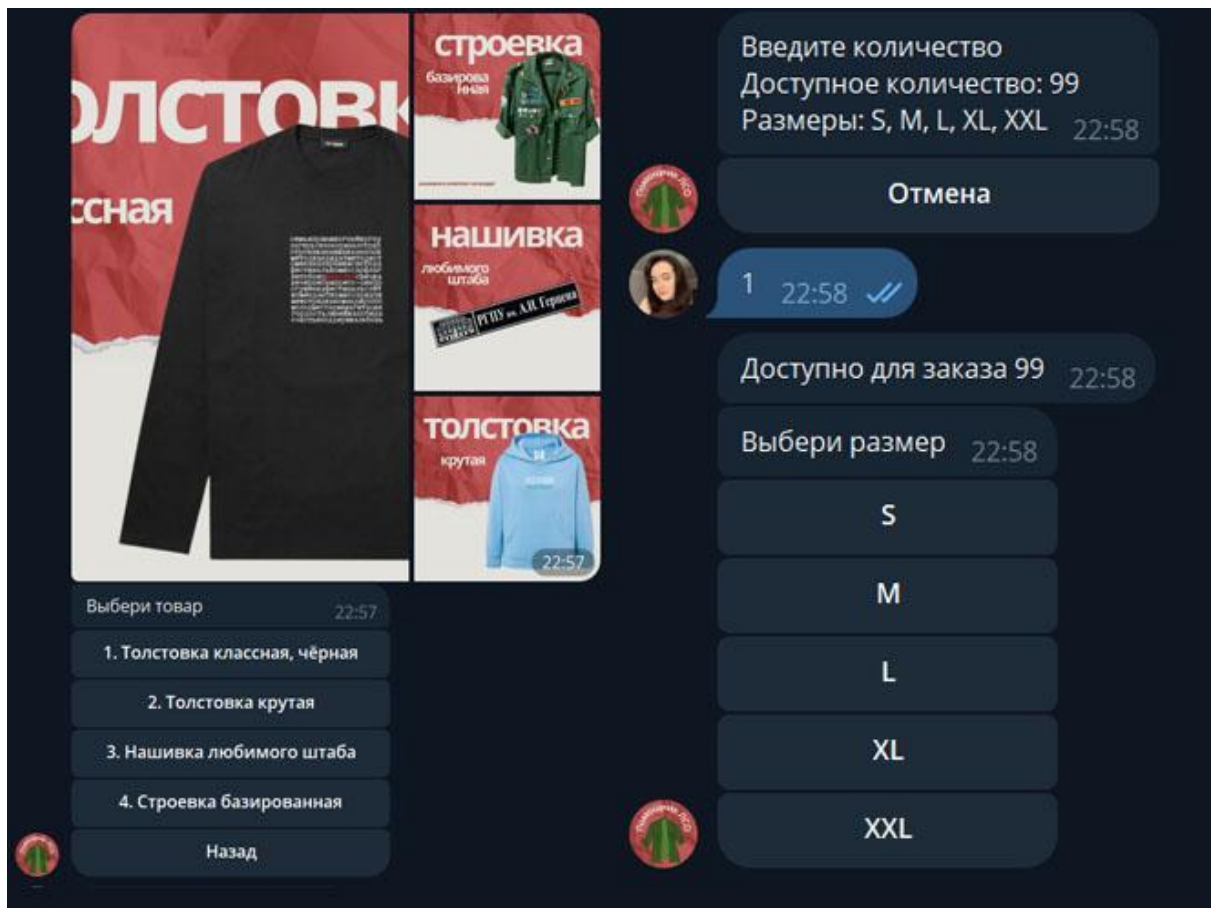


Рисунок 37. Результат работы функции «Заказ атрибутики»

Для разработки хендлера функции «Отправка фото- и видеоматериалов» была использована библиотека-клиент REST API Яндекс.Диска «YaDisk». Для загрузки материалов на диск, создается объект YaDisk для работы с Яндекс.Диском, используя токен доступа, полученный из переменной окружения. При получении файла или группы файлов бот проверяет тип файла: фото, видео или документ. Для загрузки файла, сначала бот получает id файла от API Телеграма. Далее создается экземпляр объекта BytesIO для хранения файла и загрузка файла из Telegram в объект photo [15]. В конце объект photo загружается на Диск в указанную папку. На рисунке 38 приведён полный код процесса загрузки файлов.

```

photo_info = await bot.get_file(file_id)
photo = BytesIO()
await bot.download_file(photo_info.file_path, destination=photo)
try:
    if not y.exists(f'/{file_path}'): # Проверка существования папки
        y.mkdir(f'/{file_path}') # Создание папки
    if not y.exists(f'/{file_path}/{file_name}'): # Проверка существования файла
        y.upload(photo, dst_path: f'/{file_path}/{file_name}') # Загрузка файла
    await message.answer('Файл загружен успешно!')
except:
    await message.answer('При загрузке файла произошла ошибка!')

```

Рисунок 38. Процесс загрузки файлов на Яндекс.Диск

Результат работы функции «Отправка фото- и видеоматериалов» представлен на рисунке 39.

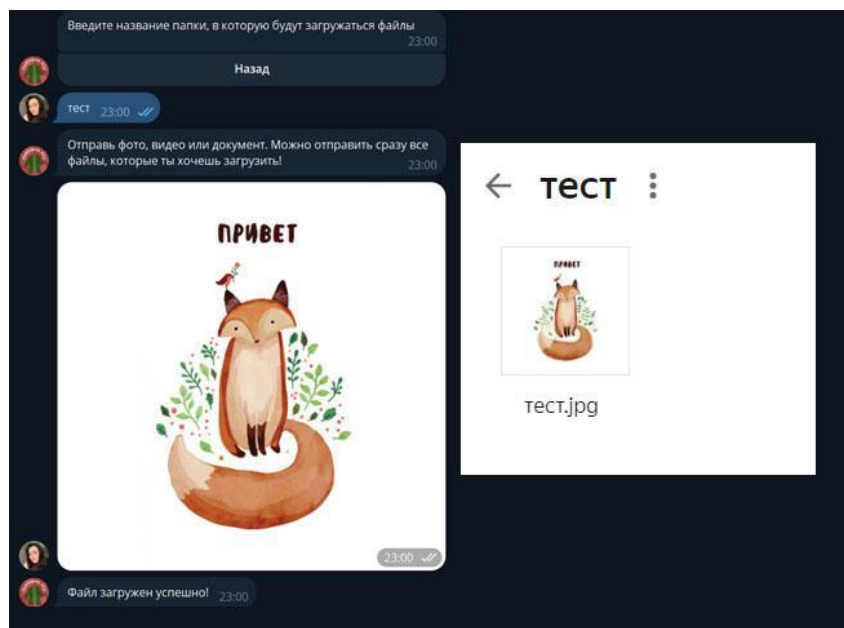


Рисунок 39. Результат работы функции «Отправка фото- и видеоматериалов»

Последняя функция бота – кнопка с ссылкой на приложение «Я боец», так как его использование актуально для бойцов линейных студенческих отрядов Санкт-Петербурга. Код реализации этого хендлера представлен на рисунке 40.

```

@user_router.callback_query(F.data == "app_lso")
async def app_lso_func(call: CallbackQuery, state: FSMContext):
    await call.message.edit_text(text="Нажимая на кнопку, ты перейдешь в ВК-приложение 'Я Боец'",
                                reply_markup=await app_kb())

```

Рисунок 40. Программный код функции перехода на приложение «Я боец»

Результат работы функции «приложение «Я боец»» представлен на рисунке 41.

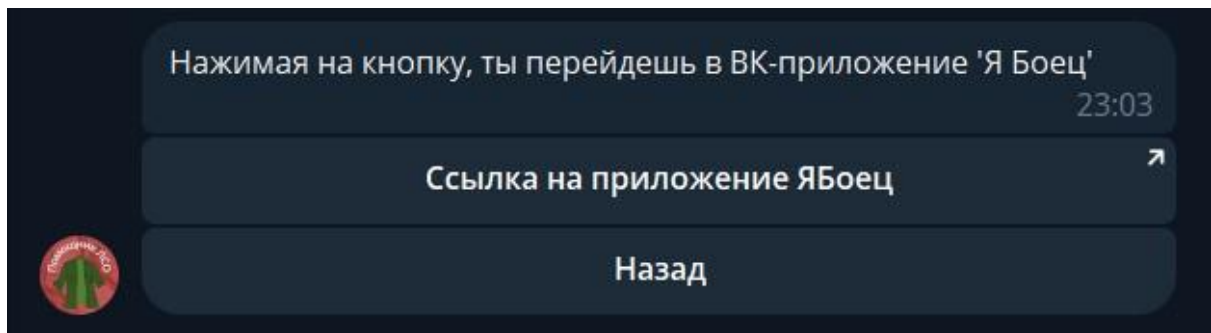


Рисунок 41. Результат работы функции «приложение «Я боец»

Разработка панели администратора

Для создания панели администратора, был установлен фреймворк Django с помощью команды `pip install django`. Далее в терминале, находясь в директории проекта бота, была выполнена команда `django-admin startproject admin_panel`, после чего была автоматически сгенерирована папка с базовой структурой проекта Django [17]. Далее, перейдя в директорию нового проекта, была введена команда `python manage.py startapp telebot`, после чего было создано само приложение панели администратора.

Основная задача при разработке панели администратора, это создания интерфейса для отображения базы данных, а также самой базы данных. При создании нового проекта Django создаёт основные файлы, в которых и происходит разработка панели администратора. Описание разработанного содержания этих файлов приведено далее.

В файле `models.py` были определены модели данных бота. Модели — это классы Python, которые представляют таблицы в базе данных. На рисунке 42 представлена абстрактная модель, поля которой добавляются к полям моделей, которые наследуются от нее.

```

class CreatedModel(models.Model):
    """Абстрактная модель."""
    created = models.DateTimeField(
        verbose_name='Дата создания',
        auto_now_add=True
    )
    updated = models.DateTimeField(
        auto_now=True,
        verbose_name="Дата изменения"
    )

    class Meta:
        abstract = True

```

Рисунок 42. Код абстрактной модели

Далее были созданы все модели для функций бота. На рисунке 43 представлена модель для функций, связанных с отправкой и поиском методических разработок.

```

class MethodsModel(models.Model):
    url = models.CharField(
        max_length=150,
        verbose_name='Ссылка на метод разработку'
    )
    name = models.CharField(
        max_length=100,
        verbose_name='Название методической разработки'
    )
    age_str = models.CharField(
        max_length=30,
        verbose_name='Возраст детей'
    )
    type_event = models.CharField(
        max_length=30,
        verbose_name='Вид мероприятия'
    )
    type_develop = models.CharField(
        max_length=30,
        verbose_name='Тип разработки'
    )

```

Рисунок 43. Модель для функции, связанных с отправкой и поиском методических разработок

В файле admin.py были зарегистрированы модели Django и настроен их внешний вид. На рисунке 44 показан пример регистрации и настройки отображения модели для функций, связанных с отправкой и поиском методических разработок.

```
@admin.register(*models: MethodsModel, site=bot_admin)
class MethodsAdmin(admin.ModelAdmin):
    list_display = (
        'name',
        'url',
        "age_str",
        'type_event',
        "type_develop",
    )
```

Рисунок 44. пример регистрации и настройки отображения модели

В файле `apps.py` была создана конфигурация для панели Django (рисунок 45). Конфигурация устанавливает имя приложения, удобочитаемое имя и тип поля для первичных ключей. Так как каждая модель в Django должна иметь первичный ключ, который идентифицирует каждую запись в базе данных, `default_auto_field = 'django.db.models.BigAutoField'` – это настройка, обеспечивающая автоматическое создание уникальных идентификаторов для каждой записи в базе данных.

```
from django.apps import AppConfig

class TelebotConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'admin_panel.telebot'
    verbose_name = 'Управление ботом'
```

Рисунок 45. Конфигурация для панели Django

В файле `admin_panel.urls.py` был реализован процесс перенаправления всех запросов, относящихся к боту, на `admin_panel.telebot`, где и описываются все параметры панели администратора, в частности – базы данных.

Администратор создается как суперюзер (англ. *superuser*) на моменте запуска проекта django в основной файле запуска проекта `bot.py` (рисунок 46). Логин и пароль содержатся в файле `.env`.

```
from tgbot.models.db_commands import BaseService
await BaseService.create_super_user(config.misc.super_user_name, config.misc.super_user_pass)
```

Рисунок 46. Создание суперюзера при запуске бота

На рисунке 47 представлен скриншот панели администратора.

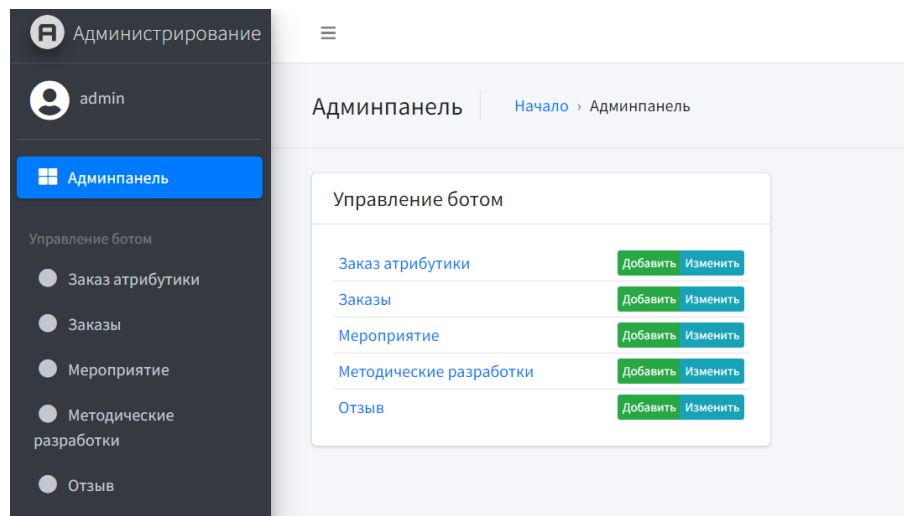


Рисунок 47. Скриншот панели администратора

Размещение готового Телеграмм-бота на сервере

Для размещения программного продукта на сервере была применена такая технология как Docker. Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации [18]. То есть задача «упаковать» все сервисы продукта в контейнеры для загрузки на сервер или в любой другой изолированной среде. После скачивания и установки Docker был создан файл Dockerfile, который показан на рисунке 48. В этом файле были прописаны команды для разворачивания нужной для запуска вашего бота среды.

```
FROM python:3.10-buster
ENV BOT_NAME=$BOT_NAME

WORKDIR /usr/src/app/"${BOT_NAME:-tg_bot}"

COPY requirements.txt /usr/src/app/"${BOT_NAME:-tg_bot}"
RUN pip install -r /usr/src/app/"${BOT_NAME:-tg_bot}"/requirements.txt
COPY . /usr/src/app/"${BOT_NAME:-tg_bot}"
```

Рисунок 48. Программный код файла Dockerfile

Далее был создан файл `docker-compose.yml`, в котором было описано, из каких сервисов (контейнеров) состоит программный продукт, как они связаны друг с другом, какие настройки и переменные окружения им нужны. На рисунке 49 представлены описания контейнеров для бота и панели администратора.

```

django:
  container_name: admin_panel
  build:
    context: .
  command: >
    sh -c "
      python django_app.py migrate &&
      python django_app.py runserver 0.0.0.0:80 --noreload"
  ports:
    - "80:80"
  networks:
    - tg_bot
  restart: always
  env_file:
    - ".env"
  depends_on:
    - db
  volumes:
    - ./src

bot:
  image: "${BOT_IMAGE_NAME:-tg_bot-image}"
  container_name: "${BOT_CONTAINER_NAME:-tg_bot-container}"
  stop_signal: SIGINT
  build:
    context: .
  working_dir: "/usr/src/app/${BOT_NAME:-tg_bot}"
  volumes:
    - ./usr/src/app/${BOT_NAME:-tg_bot}
  command: python3 -m bot
  restart: always
  env_file:
    - ".env"
  networks:
    - tg_bot
  depends_on:
    - db
    - django
    - redis

```

Рисунок 49. Описание контейнеров бота и панели администратора

После подготовки всех контейнеров, а также файла `requirements.txt`, где были перечислены все используемые библиотеки и пакеты, необходимые для работы проекта, происходит выгрузка программы на сервер. Для этого необходимо было подключиться к серверу по сетевому протоколу SSH. На сервере была создана папка `bot`, куда были перенесены все файлы программного продукта. Далее в терминале на сервере были применены миграции ранее созданных моделей и выполнена команда `docker-compose up --build`, которая используется для запуска приложения, описанного в файле `docker-compose.yml`, с предварительной сборкой образов Docker.

Планы по развитию проекта

Представленный проект в настоящей выпускной квалификационной работе, а именно – панель администратора, является неким прототипом. В проект планируется внести множество изменений по добавлению нового функционала в панель администратора, а именно:

1. Внедрить систему ролей администратора для каждого руководителя из командного состава;
2. Добавить возможность выборочного изменения внесенных данных пользователем, например для функции «Загрузка методических разработок»;
3. Добавить возможность сортировки и выгрузки данных из базы данных, отображающихся в панели администратора;
4. Добавить возможность загружать фотографии в существующие папки на Яндекс Диске, реализовывая это с помощью inline-клавиатуры;
5. Добавить возможность выбора пользователя в функции «Письма на сезон» вместо ручного введения данных.

Также список функционала будет пополняться в процессе дальнейшей работы над продуктом.

ЗАКЛЮЧЕНИЕ

Цель данной выпускной квалификационной работы заключалась в разработке Telegram-бота для организации работы Линейного студенческого отряда.

Для достижения этой цели были решены следующие задачи:

- проведен анализ развития отрядного движения на примере Штаба Студенческих отрядов РГПУ им. А. И. Герцена;
- проведен анализ существующих программных продуктов для студенческих отрядов;
- проведен обзор существующих инструментов и информационных технологий для создания Telegram-бота;
- рассмотрены особенности создаваемого Telegram-бота и балы разработана структура;
- разработан Telegram-бот с определенными особенностями;
- разработана административная панель и база данных для записи передаваемой пользователем информации;
- программный продукт развернут на сервере.

Таким образом, все поставленные задачи были успешно выполнены, а цель была достигнута. По итогам данной работы был разработан Telegram-бота «Помощник ЛСО».

СПИСОК ЛИТЕРАТУРЫ

1. Положение "Устав Молодежной общероссийской общественной организации «Российские Студенческие Отряды»" от 12.03.2022 // трудкрут.рф. - с изм. и допол. в ред. от 12.03.2022.
2. Положение "Положение о линейном студенческом отряде Санкт-Петербургского регионального отделения молодежной общероссийской общественной организации «Российские студенческие отряды»" от 19.02.2024 // so.spb.ru. - с изм. и допол. в ред. от 19.02.2024.
3. Аудитория Telegram. Отчет по данным: Январь 2024 // Mediascope URL: mediascope.net/ (дата обращения: 02.05.2023).
4. ТIOBE: Index for May 2024 // TIOBE URL: <https://www.tiobe.com/tiobe-index/> (дата обращения: 04.05.2023).
5. Рытикова В. О. Статистика и анализ языков программирования для разработки чат-бота в Telegram // Весенние дни науки: Информационно-математические технологии в социальных и экономических системах. - Ростов-на-Дону: УрФУ, 2020. - С. 741-746.
6. Какую библиотеку на Python выбрать для создания Телеграм-бота // Хабр URL: <https://habr.com/ru/companies/otus/articles/771110/> (дата обращения: 06.05.2023).
7. Шмелев А.Н., Жилина Е.В. Разработка Telegram-бота для новостных сайтов // Информационные системы, экономика и управление. - Ростов-на-Дону: Издательско-полиграфический комплекс РГЭУ (РИНХ), 2022. - С. 107-113.
8. Филатова З. М., Закирова Н. Р. Создание Telegram-бота для автоматизации административной деятельности // Проблемы современного педагогического образования. 2023. №79-4.
9. Гугаев М. В. Разработка Telegram-бота и портала для информационной поддержки студентов: дис. канд. т.н. наук: 09.04.01. - Москва, 2019 с.
10. А.П. Плетухин, М.А. Польская, А.А. Плетухина, О.Л. Серветник ORM как альтернативный способ общения с СУБД // Инновационные тренды и

драйверы устойчивого развития социальноэкономических процессов в условиях перехода к цифровому обществу: сборник научных статей по материалам Международной научно-практической конференции, Ставрополь, 16–17 октября 2023 года. - Ставрополь: Ставропольский государственный аграрный университет, 2023. - С. 591-595.

- 11.Строенков, А. А. Организация доступа к реляционной базе данных на основе технологии ORM с использованием паттерна Active Record // Молодой ученый. — 2018. — № 20 (206). — С. 31-35. — URL: <https://moluch.ru/archive/206/50571/> (дата обращения: 03.05.2024).
- 12.Корчилава, А.В., Носова, Е.И. База данных по истории и технологии бумаги: состав, структура, особенности // Культура и технологии. 2023. Том 8. Вып. 1. С. 34-49.
- 13.Молодяков С.А., Милицын А.В. Алгоритмы работы с мультимедийными данными в Telegram-боте. - 1-е изд. - СПб.: Издательство Политехнического университета, 2024. - 587 с.
- 14.Что такое бот в Telegram: виды и функции // GeekBrains URL: <https://gb.ru/blog/chto-takoe-telegram/#2> (дата обращения: 10.05.2023).
- 15.Кондратьев А.В. Разработке Telegram-ботов на языке Python с использованием фреймворка aiogram 3.x. - 3-е изд. - М.: Электронное издание, 2022. - 287 с.
- 16.Пушкарева Д. П. Телеграм-бот для организаций предзаказов в сети кофеен // Международный журнал гуманитарных и гуманитарных наук. 2023. №6-3 (81).
- 17.Солиев Б. Н. Путеводитель по построению веб-API на Django Шаг за шагом с Django REST framework — от моделей до проверки работоспособности // Al-Farg’oniy avlodlari. 2023. №4.
- 18.Чиганов Д.Р. Докер: ключ к контейнеризации и масштабируемости// Вестник науки. М.: 2023. №7 (64).