МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

*Вычислительной техники*

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(полное название кафедры)

# Утверждаю

#### Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, инициалы, фамилия)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

*Якименко А.А.*

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

*Павлова Ивана Алексеевича*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество студента – автора работы)

*Разработка Telegram-бота BMW Club Новосибирск*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(тема работы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Факультет автоматики и вычислительной техники*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(полное название факультета)

*09.03.04 Программная инженерия*

Направление подготовки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(код и наименование направления подготовки бакалавра)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Старший преподаватель*

*Павлов Иван Алексеевич*

### Руководитель

### от НГТУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ученая степень, ученое звание)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

### Автор выпускной квалификационной работы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(факультет, группа)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

*АВТФ, АВТ-943*

*Мищенко Полина Валерьевна*

Новосибирск 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

*Вычислительной техники*

Кафедра \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(полное название кафедры)

##### УТВЕРЖДАЮ

# Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

*Якименко А.А.*

*Введите здесь свой текст*

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

*Павлову Ивану Алексеевичу*

студенту *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(фамилия, имя, отчество)

*09.03.04 Программная инженерия*

Направление подготовки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(код и наименование направления подготовки бакалавра)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Факультет автоматики и вычислительной техники*

(полное название факультета)

*Разработка Telegram-бота BMW Club Новосибирск*

Тема \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(полное название темы выпускной квалификационной работы бакалавра)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Разработка Telegram-бота для автоматизации процесса администрирования, информирования и обеспечения обратной связи с участниками автоклуба.*

Исходные данные (или цель работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Введение*

*Постановка задачи*

1. *Анализ аналогичных программных решений*
2. *Техническое задание*
3. *Обоснование выбора средств разработки*
4. *Архитектура приложения*

Структурные части работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*5. Сценарии взаимодействия с системой*

*6. Разработка бота*

*7. Тестирование и апробация*

*Заключение*

*Список литературы*

*Приложения*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Задание согласовано и принято к исполнению.

### Руководитель

### от НГТУ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ученая степень, ученое звание)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

### Студент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(факультет, группа)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

*Мищенко Полина Валерьевна*

*Павлов Иван Алексеевич*

*АВТФ, АВТ-943*

*Старший преподаватель*

Тема утверждена приказом по НГТУ № 1072/2 от 06.03.2023 г.

изменена приказом по НГТУ № \_\_\_\_\_\_\_\_\_ от «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 202\_\_г.

ВКР сдана в ГЭК № 11.1 , тема сверена с данными приказа

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата)

Харюткина София Александровна

(фамилия, имя, отчество секретаря государственной

экзаменационной комиссии по защите ВКР)

**РЕФЕРАТ**

Выпускная квалификационная работа выполнена студентом Павловым Иваном Алексеевичем.

Место создания – ФГБОУ ВО «Новосибирский государственный технический университет».

Руководитель – старший преподаватель кафедры Вычислительной техники, Мищенко Полина Валерьевна.

Пояснительная записка состоит из 7 глав, содержит 75 страницы основного текста, 60 рисунков, 7 источников и 2 приложения. Полный объём пояснительной записки составляет 130 страницы.

Список ключевых слов: Python, AIOgram, Telegram-бот, асинхронность, MVC, PostgreSQL.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc138326073)

[ПОСТАНОВКА ЗАДАЧИ 9](#_Toc138326074)

[1 АНАЛИЗ АНАЛОГИЧНЫХ ПРОГРАММНЫХ ПРОДУКТОВ 10](#_Toc138326075)

[1.1 Censure\_bot 10](#_Toc138326076)

[1.2 Birthday Bot 11](#_Toc138326077)

[2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 12](#_Toc138326078)

[2.1 Введение 12](#_Toc138326079)

[2.1.1 Общие сведения 12](#_Toc138326080)

[2.1.2 Определения, акронимы и сокращения 12](#_Toc138326081)

[2.2 Назначение разработки 13](#_Toc138326082)

[2.3 Необходимая функциональность 13](#_Toc138326083)

[2.3.1 Регистрация 13](#_Toc138326084)

[2.3.2 Система разного доступа и разграничения полномочий 14](#_Toc138326085)

[2.3.3 Функция широковещательной рассылки 16](#_Toc138326086)

[2.3.4 Система оповещений 16](#_Toc138326087)

[2.3.5 Система Анти-мат 17](#_Toc138326088)

[2.3.6 Система клубных мероприятий 17](#_Toc138326089)

[2.3.7 Система оповещения администраторов 18](#_Toc138326090)

[2.3.8 Взаимодействие с информацией о пользователях 18](#_Toc138326091)

[2.4 Требования к программному обеспечению 19](#_Toc138326092)

[2.4.1 Общие требования 19](#_Toc138326093)

[2.4.2 Требования к графическому интерфейсу 19](#_Toc138326094)

[2.4.3 Требования к взаимодействию с базой данных 20](#_Toc138326095)

[2.4.4 Системные требования к размещению бота 20](#_Toc138326096)

[3 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ 22](#_Toc138326097)

[3.1 Язык программирования и платформа 22](#_Toc138326098)

[3.1.1 Python 22](#_Toc138326099)

[3.1.2 Java 24](#_Toc138326100)

[3.1.3 PHP 25](#_Toc138326101)

[3.1.4 Сравнение языков 27](#_Toc138326102)

[3.2 Основной Python фреймворк 29](#_Toc138326103)

[3.3 СУБД 31](#_Toc138326104)

[4 АРХИТЕКТУРА ПРИЛОЖЕНИЯ 34](#_Toc138326105)

[4.1 Общая архитектура приложения 34](#_Toc138326106)

[4.2 Режим работы бота 35](#_Toc138326107)

[5 СЦЕНАРИИ ВЗАИМОДЕЙСТВИЯ С СИСТЕМОЙ 37](#_Toc138326108)

[6 РАЗРАБОТКА БОТА 39](#_Toc138326109)

[6.1 Структура бота 39](#_Toc138326110)

[6.2 База данных 43](#_Toc138326111)

[6.3 Визуальная часть бота 47](#_Toc138326112)

[6.4 Дополнительные функции 48](#_Toc138326113)

[7 ТЕСТИРОВАНИЕ И АПРОБАЦИЯ 50](#_Toc138326114)

[7.1 Ручное тестирование 50](#_Toc138326115)

[7.1.1 Незарегистрированный пользователь 50](#_Toc138326116)

[7.1.2 Участник клуба (обычный пользователь) 58](#_Toc138326117)

[7.1.3 Пользователь+ 62](#_Toc138326118)

[7.1.4 Администратор 64](#_Toc138326119)

[7.1.5 Владелец бота 72](#_Toc138326120)

[7.2 Апробация 73](#_Toc138326121)

[ЗАКЛЮЧЕНИЕ 74](#_Toc138326122)

[СПИСОК ЛИТЕРАТУРЫ 75](#_Toc138326123)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ 76](#_Toc138326124)

[ПРИЛОЖЕНИЕ Б. АКТ О ВНЕДРЕНИИ РЕЗУЛЬТАТОВ ВКРБ 130](#_Toc138326125)

ВВЕДЕНИЕ

В современной жизни людей всё большую роль играют мессенджеры - клиентские программы, необходимые для мгновенного обмена сообщениями.

Первый мессенджер, который получил широкое распространение и стал популярным, был ICQ. Он был разработан израильской компанией Mirabilis и выпущен 15 ноября 1996 года и предлагал возможность обмена сообщениями в реальном времени между пользователями через Интернет.

Изначально создававшиеся как программы, решающие лишь одну или несколько задач, а именно: получение и доставка сообщений, мессенджеры позволяют поддерживать постоянную связь с коллегами, друзьями или людьми по интересам. Однако с развитием Интернета и большей популяризацией мессенджеров наиболее актуальной стала проблема эффективного и простого управления группами.

К сожалению, далеко не вся встроенная функциональность мессенджеров помогает решить определенный ряд задач, связанных с автоматизацией процесса администрирования.

Решением данной проблемы было добавление возможности самим пользователям создавать и определять те задачи и методы их решения, которые подходили бы именно им, с помощью небольших программ, которые могли напрямую функционировать в системе мессенджера - ботов.

Одним из наиболее популярных мессенджеров в настоящее время является Telegram, который предоставляет широкие возможности для разработки собственных ботов. Telegram-боты могут выполнять различные задачи: от предоставления информации и развлечений до автоматизации бизнес-процессов.

Одному из сообществ в Новосибирске, посвященной автомобильной тематике и имеющую крупную группу в Telegram, «BMW Club Новосибирск», понадобилась автоматизация процесса сбора информации и администрирования группы в Telegram. Согласно основным потребностям заказчика, бот должен быть разработан с помощью языка программирования Python, обеспечен бесперебойной работой на сервере, базой данных и различными уровнями доступа.

Выпускная квалификационная работа бакалавра посвящена разработке программы для автоматизации процессов администрирования группы и решения задач организационного характера, заказчиком которого является сообщество «BMW Club Новосибирск». Разработанная программа должна решить проблему заказчика со сбором актуальных данных, цензурирования группы, оповещений, организации и формализации событий.

ПОСТАНОВКА ЗАДАЧИ

Целью данной выпускной квалификационной работы является разработка Telegram-бота для автоматизации процесса администрирования, информирования и обеспечения обратной связи с участниками автоклуба.

Для достижения цели были поставлены следующие задачи:

* Проанализировать аналогичные программные продукты;
* Провести обоснование выбора средств разработки;
* Определить требования к боту;
* Спроектировать архитектуру бота;
* Определить схему взаимодействия пользователя и бота
* Разработать структуру бота;
* Разработать базу данных и компоненты взаимодействия базы данных с приложением;
* Реализовать основные функциональные возможности бота;
* Провести тестирование и апробацию бота.

1 АНАЛИЗ АНАЛОГИЧНЫХ ПРОГРАММНЫХ ПРОДУКТОВ

Большинство из представленных в открытом доступе ботов не имеют функциональности, той которая подошла бы заказчику, в качестве аналогичного программного продукта будут приводится в пример боты, имеющие схожую функциональную часть или решающую аналогичную проблематику.

1.1 Censure\_bot

Это бот был написан для борьбы с нецензурными выражениями (матом) в группах. Удаляет сообщения и ограничивает нарушителя. Реагирует на мат в текстах сообщений и никах пользователей. Был создан 18 октября 2020 года.

**Установка**

Установка осуществляется через добавление бота в группу и после того как участник этой группы произнесет нецензурное слово, бот должен среагировать. Для удаления и блокировки сообщений и участников необходимо назначить бота администратором группы с нужными правами.

**Настройка**

Для настройки необходимо отправить команду /settings@censure\_bot (в самой группе) и следуйте инструкции. По умолчанию сообщение с нецензурным словом удаляется, отправка сообщений блокируется ("mute") по формуле [количество нарушений] ² (часов) (первое нарушение 1 час, второе 4 часа и т.д).

**Алгоритм обнаружения нецензурной лексики**

Нецензурная лексика данным ботом обнаруживается исходя из того, что большинство нецензурных слов в базе на русском языке, образованы от четырёх общепринятых матов. Возможны ложные срабатывания (администратор старается отслеживать и добавлять в исключения). Важно понимать, что нарушители могут исковеркать нецензурное слово, некоторые варианты маскировки бот распознаёт, но все не учесть.

Такой метод фильтрации, весьма подходит и может вполне послужить для того, что лечь в основу системы анти-мат, особенно функция ограничения пользователя.

1.2 Birthday Bot

Этот бот был создан для того, чтобы люди могли поздравлять людей с днем рождения в случае, если пользователь сам знает у кого, когда день рождение, то он добавляет его в базу к боту.

**Основные команды**

Все команды должны начинаться с символа «/».

/start - Запустить бота

/add - Добавить день рождение

/when\_my\_birthday - Когда у меня день рождение?

/change - Изменить дату рождения

**Установка**

Бот не требует установки т.к. общается в частном чате и в inline режиме.

**Настройка**

Для того чтобы настроить бота необходимо создать некую базу из дней рождения тех людей, которые пользователю уже известны и бот в нужный момент за день напоминает об этом.

В целом такой подход к поздравлению пользователей заказчику не подходит так как ему необходимо чтобы люди в ограниченной системе при регистрации указывали день рождения и в момент, когда необходимо напомнить бот делает это, причем сама функция должна быть доступна ограниченному кругу лиц.

2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

В соответствии с требованиями заказчика было разработано техническое задание на создание нового программного продукта. Данный продукт предназначен для решения задач, связанных с автоматизацией организационных процессов в сообществе. Он должен обладать высокой производительностью, надежностью и функциональностью, а также быть удобным в использовании для конечных пользователей. В данном разделе описаны все требования и условия, которые необходимо учесть при создании данного программного продукта.

2.1 Введение

2.1.1 Общие сведения

**Полное наименование работ:** Разработка Telegram-бота BMW Club Новосибирск.

**Заказчик:** ООО «Альтернатива»

**Исполнитель:** Павлов И.А.

2.1.2 Определения, акронимы и сокращения

ПО – программное обеспечение.

БД – база данных.

СУБД – система управления базами данных.

Telegram-бот (Telegram bot, бот) – это программа, созданная для автоматизированной обработки текстовых сообщений от пользователей и предоставления ответов или выполнения действий, которые необходимы для решения поставленных задач, в мессенджере Telegram.

Графический интерфейс (ГИ, GUI, Graphical User Interface) – это способ представления информации пользователю с помощью графических элементов, таких как окна, кнопки, поля ввода, меню, иконки. Предполагает замену текстового интерфейса на тот, где пользователи могут использовать мышь, клавиатуру или сенсорный экран для взаимодействия с приложением.

Встроенная-кнопка (Inline Button) – элемент пользовательского интерфейса, который закреплен за определенным сообщением, отправленным ботом, и является активной кнопкой, при нажатии на которую со стороны пользователя не отправляется текст в чат с ботом.

Клавиатурная кнопка (Keyboard Button) - элемент пользовательского интерфейса, кнопка, которая отображается у пользователя вместо стандартной клавиатуры и представляет собой меню, при выборе пункта которого пользователь отправляет сообщение боту в чат, на которое чат-бот реагирует.

2.2 Назначение разработки

Основным назначением разрабатываемого ПО является применение его сообществом «BMW Club Новосибирск» для автоматизации сбора и систематизации информации от участников клуба и администрирования Telegram группы.

2.3 Необходимая функциональность

2.3.1 Регистрация

Суть регистрации пользователя заключается в том, чтобы получить определенную информацию и записать её в базу данных , а именно:

* Фамилия Имя Отчество
* Дата рождения
* Номер телефона
* Род деятельности/ Профессия
* Фото автомобиля
* Государственный номер автомобиля
* Сбор информации о том может ли участник клуба стать партнером

Приоритет при реализации этого функционала должен отдаваться тому, чтобы пользователю нужно было как можно меньше вручную указывать эту информацию, она может быть, записанная в базу данных или же может быть реализован реестр в локальном хранилище в виде файла.

До регистрации бот должен максимально подробно объяснить пользователю что и по какой причине запрашивается, расписать каждый этап заполнения данных. В процессе регистрации должны быть доступны кнопки перерегистрации при ошибке заполнения данных и обращения в поддержку в случае, если пользователь не уверен в правильности понимания запрашиваемой информации.

Так же до регистрации у пользователя бота должен быть выбор того, как зарегистрироваться, а именно должно быть 3 пункта:

1. Я бизнес-партнер
2. Я рекламодатель
3. Я участник клуба

Первые два пункта являются регистрацией обращения к администраторам и должен запрашивать минимальное количество информации, а именно: ФИО, номер телефона и указанием информации о том с чем он обращается, например, какой вид услуги может предложить или хотел бы получить и т.д., так как предполагается, что данный пользователь может и не являться участником клуба, но имеет цель стать бизнес-партнером клуба или прорекламировать свою продукцию или услуги.

При регистрации участника, рекламодателя или бизнес-партнера администраторы должны быть оповещены об этом и иметь возможность связаться с каждым пользователем лично.

2.3.2 Система разного доступа и разграничения полномочий

Предполагается, что у каждого пользователя будет атрибут, что будет определять уровень его доступа к функциональности бота. Подразумевается, что для каждой роли в системе будет продуман свой интерфейс.

Виды прав доступа:

1. Владелец бота
2. Администратор
3. Пользователь с расширенной функциональностью
4. Обычный пользователь

Описание каждой роли и её функционал:

1. Владелец бота: это самая уполномоченная роль, основная её особенность заключается в том, что может снимать и назначать администраторов в специальной группе, в остальном ему будет доступна та же самая функциональность, что и администратору.
2. Администратор: доверенное лицо владельца бота, назначается им лично в специальной группе. Доступна следующая функциональность:
   1. Доступ к дням рождения всех, зарегистрированных в боте, участников клуба и уведомлениях о ближайших.
   2. Подробная инструкция пользования всей функциональности бота
   3. Получения информации о всех и конкретных, зарегистрированных в боте, участников клуба, в том числе в виде excel файла.
   4. Доступ к управлению системой мероприятий клуба, а именно: просмотр, добавление, удаления и редактирование мероприятий
   5. Доступ к системе широковещательной рассылки сообщений, которая через бот должна присылать каждому зарегистрированному пользователю объявление в личные сообщения.
   6. Изменение информации конкретного зарегистрированного пользователя
   7. Назначение и снятие пользователей с расширенной функциональностью в специальной группе
   8. Возможность отвечать в личных сообщениях на обращения пользователей с вопросами в специальной группе
   9. Возможность лично обращаться с только зарегистрировавшимися пользователями с помощью уведомлений в специальной группе
3. Пользователь с расширенной функциональностью: далее будет называться просто пользователь+. Является активной частью сообщества и назначается администратором или владельцем бота. Доступна следующая функциональность:
   1. Доступ к дням рождения учеников клуба и уведомлениям о ближайших днях рождения
   2. Возможность в любой момент обратится в поддержку к админам по возникшему вопросу
   3. Доступ к мероприятиям клуба, а именно: Просмотр ближайших мероприятий, возможность подписаться или отписаться от участия в мероприятии и уведомления о завтрашнем мероприятии
4. Обычный пользователь: состояние зарегистрировавшегося участника клуба по умолчанию. Отличие от пользователя с расширенной функциональностью в том, что не имеет никакого доступа к личной информации о других участниках клуба, в том числе днях рождения, в остальном также имеет доступ к поддержке и системе мероприятий клуба.

2.3.3 Функция широковещательной рассылки

Данная функция должна быть доступна к использованию только людьми роли Администратор и Владелец бота. Должна использоваться для донесения важной информации для всех зарегистрированных пользователей бота. Предполагается автоматическая модерация нецензурной лексики и закрепления этого сообщения в чате каждого пользователя.

2.3.4 Система оповещений

Данная система будет по желанию пользователя уведомлять пользователя о том, что приближается день рождения, одного из участников сообщества или день проведения запланированного мероприятия. Напоминания о днях рождениях, должны приходить за день в 11:00, о мероприятиях за день в 13:00. Также должна быть предусмотрена функция отключения и включения оповещений, по умолчанию все уведомления будут включены.

2.3.5 Система Анти-мат

Данная система предполагает определение в группе нецензурной лексики и цензурирование сообщений и их удаление в автоматическом режиме.

Также предусмотрено ограничения доступа к отправке сообщений в группу в случае фиксации более 3х случаев употребления ненормативной лексики за неделю. Ограничение может быть отменено администратором группы в настройках группы.

2.3.6 Система клубных мероприятий

Доступ к этой системе будет у всех пользователей, но к разной её части. Полнофункциональный доступ будут иметь: Администратор и Владелец бота, частичный: Пользователь+ и Обычный пользователь.

Полнофункциональный доступ предполагает:

1. Создание мероприятия:
   1. Ввод названия мероприятия
   2. Описание мероприятия
   3. Описание места проведения
   4. Выбор даты проведения мероприятия (Специальный календарный интерфейс выбора даты)
   5. Ввод времени начала мероприятия
   6. Подтверждение правильности введенных данных
2. Список всех созданных мероприятий:
   1. Просмотр всех мероприятий
   2. Редактирование конкретного мероприятия
   3. Удаление конкретного мероприятия

Частичный доступ предполагает:

1. Просмотр всех актуальных мероприятий, то есть тех мероприятий дата проведения, которого ещё не прошла.
2. Возможность подписаться или отписаться на конкретное мероприятие, что дает возможность организатору мероприятия и самому участнику понять сколько ориентировочно, человек должно прийти.

2.3.7 Система оповещения администраторов

Суть данной системы заключается в том, что должна быть супергруппа, особенность которой это деление основного чата группы на темы, в этой группе должны быть следующие темы:

* Главная: в этой теме должны сниматься и назначаться администраторы и пользователи с расширенной функциональностью с помощью команд
* Поддержка: в этой теме должны приходить вопросы от пользователей бота, с указанием пользователя, самого вопроса, и возможностью удалить вопрос во избежание повторных ответов со стороны других админов
* Уведомления о партнерах: эта тема нужна для того, чтобы администраторы могли увидеть предварительную информацию о партнере и связаться с ним для обсуждения деталей сотрудничества
* Уведомления о рекламодателях: предназначение этой темы аналогично предыдущей, но для рекламодателей
* Уведомления об участниках: в эту тему приходят уведомления о новых участниках клуба, администраторы связываются с ними для проверки и добавления в группу

2.3.8 Взаимодействие с информацией о пользователях

Получать информацию о зарегистрированных пользователях могут только Администраторы и Владелец Бота.

Получение информации должно быть реализовано через указание:

1. Фамилию Имя Отчество, должна быть предусмотрена возможность поиска по одному или нескольким компонентам ФИО
2. Государственному номеру автомобиля
3. Получение данных о всех пользователях, ботом должен формироваться excel файл с содержанием следующей информацией о всех пользователях: "Фамилия", "Имя", "Отчество", "День рождения", "Номер телефона", "Род деятельности", "Информация о партнере", "Является админом", "Является пользователем+", "Гос.номер"

При получении информации о конкретном пользователе в первых двух пунктах, бот должен структурировать её и вывести в личный чат с администратором.

Редактирование информации о пользователе должно быть предусмотрена с помощью поиска по ФИО и нужна для случаев ошибочного указание информации пользователя о себе.

2.4 Требования к программному обеспечению

2.4.1 Общие требования

1. Программное обеспечение должно разрабатываться с учётом возможности дальнейшего масштабирования его функциональных возможностей.
2. Для хранения и дальнейшего использования информации в качестве системы управления базами данных должна быть внедрена встроенная СУБД.
3. ПО должно предоставлять возможность настройки администратором календарных событий, взаимодействия с БД, регистрацию пользователей

2.4.2 Требования к графическому интерфейсу

1. При запуске бота во встроенном интерфейсе Telegram должна быть предоставлена возможность зарегистрироваться
2. После регистрации пользователю должны отображаться следующие клавиатурные кнопки: «Дни рождения», «Мероприятия клуба», «Помощь»
3. В пункте «Дни рождения» должны быть доступны следующие подпункты, являющиеся клавиатурными кнопками: «Ближайшие 14 дней», «Вкл/Выкл напоминания», «Поиск по ФИО»
4. При выборе пункта «Ближайшие 14 дней» бот должен предоставить список с ФИО, датой рождения и ссылкой на профили тех людей, у которых в ближайшие 14 дней будет день рождение. Данный список является сообщением с прикрепленными встроенными-кнопками взаимодействия, а именно: «Предыдущий день», «Текущая дата» и «Следующий день», при взаимодействии с кнопками «Предыдущий день» и «Следующий день», название кнопки «Текущая дата» должна сменяться на 1 день, а содержание сообщения меняться в соответствии с информацией об этом дне.
5. При выборе пункта «Мероприятия клуба» у пользователя должны появиться клавиатурные кнопки: «Ближайшие мероприятия», «Вкл/Выкл напоминания»
6. При нажатии кнопки «Ближайшие мероприятия» будут отображены списком все запланированные администратором мероприятия, каждый элемент этого списка будет содержать в себе: Название мероприятия, Локацию, Время, Описание и встроенные кнопки для переключения между мероприятиями.
7. Для администратора будет доступна клавиатурная кнопка «Рассылка», которая будет отвечать за рассылку сообщения всем пользователям бота.
8. В пункте «Мероприятия клуба» у администратора будет дополнительная клавиатурная кнопка «Создать мероприятие», а также встроенная кнопка «Редактировать» при просмотре уже созданных.
9. При создании мероприятия будет предусмотрен унифицированный выбор даты и времени начала мероприятия, которые будет реализован через массивы встроенных кнопок, обозначающих календарь выбора даты и панель выбора времени.

2.4.3 Требования к взаимодействию с базой данных

1. Пользователь должен иметь возможность получить информацию о днях рождения и мероприятиях в любое время.
2. Администратор должен иметь возможность в любой момент редактировать и создавать мероприятия.
3. У бота должен быть бесперебойный доступ к базе данных для определения обычного пользователя и администратора и разграничения их прав.

2.4.4 Системные требования к размещению бота

1. Операционная система, например, Ubuntu, CentOS
2. Веб-сервер, такой как Nginx или Apache и иметь доступ к сети.
3. Telegram требует, чтобы взаимодействие с ботами происходило через HTTPS-соединение для обеспечения безопасности. Для этого потребуется SSL-сертификат для домена. Получить бесплатный SSL-сертификат можно от сервисов, таких как Let's Encrypt.
4. Поддержка языка программирования Python
5. Оперативная память (RAM): от 256Мб до 2 Гб
6. Процессор: не менее 1Ггц.

3 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ

3.1 Язык программирования и платформа

Так как Telegram-бот — это веб-приложение, то для разработки бота следует выбивать те языки, которые лучше всего приспособлены для работы с вебом и северами, в качестве основных языков рассмотрим: Python, Java и PHP. Ниже будут рассмотрены перечисленные языки программирования и обосновано их применение.

3.1.1 Python

Python – это высокоуровневый язык программирования, ориентированный на концепции объектно-ориентированного программирования его особенностью, является динамическая типизация и интерпретируемость, наглядный пример того, как интерпретируется Python:

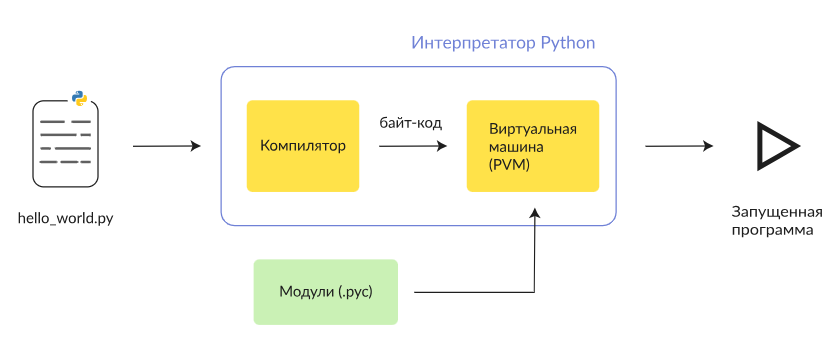


Рисунок - Поэтапное выполнение Python-программы Интерпретатором.

Сегодня это один из наиболее широко используемых языков, предлагающий простой для понимания синтаксис высокого уровня. Руководящие принципы Python заключаются в том, чтобы «упрощать все, что мы можем» и «простое лучше, чем сложное». Согласно этому опросу Stack Overflow 2020 [1], это один из самых любимых и востребованных языков.

Популярность языка, огромное количество ботов написано на Python, что как следствие дает более качественные варианты для разработки и документацию. Например, существуют популярные библиотеки, такие как python-telegram-bot, Telebot, aiogram, которые предоставляют удобные API для взаимодействия с Telegram Bot API.

Python имеет простой и понятный синтаксис, который делает его легким для изучения. Он предлагает чистый и выразительный код, что позволяет разработчикам быстро создавать итеративные прототипы и разрабатывать приложения с минимальными усилиями.

Он поддерживает различные подходы, такие как синхронный и асинхронный стиль программирования, что позволяет разрабатывать ботов с разной степенью сложности

Интеграция с другими технологиями: Python легко интегрируется с другими технологиями и сервисами. Это позволяет создавать Telegram-ботов, которые взаимодействуют с внешними API, базами данных, веб-серверами и другими системами. Python также имеет поддержку для работы с различными форматами данных, такими как JSON, XML, CSV, что делает его удобным для обработки данных, полученных от Telegram API.

К сожалению, python известен низкой скоростью. Построчное выполнение кода Python компиляторами часто влияет на время выполнения. Динамическая парадигма, которую он использует, означает, что во время выполнения должна выполняться дополнительная работа, и это также влияет на общую скорость. К примеру, для переменной в C компилятор знает тип по самому её определению. Все, что вы знаете о переменной в Python во время выполнения программы, это то, что это какой-то объект Python, простой запуск интерпретатора Python создаст множество целочисленных объектов [2].

В Python также есть поддержка асинхронности. В версии Python 3.4 был добавлен пакет asyncio, который содержит классы и функции для написания асинхронного кода. Он позволяет создавать сопрограммы или корутины – асинхронные функции и использовать событийный цикл для выполнения асинхронных операций.

Кроме того, в Python есть библиотеки, такие как aiohttp и aiofiles, которые предоставляют механизмы для работы с асинхронными операциями в сетевом вводе-выводе и файловой системе соответственно.

Также в Python есть функциональность генераторов, которые позволяют создавать итераторы с возможностью приостановки и возобновления выполнения. Это может быть использовано для написания асинхронного кода без использования asyncio.

3.1.2 Java

Java является одним из самых популярных языков программирования и широко используется для разработки масштабных и производительных приложений. Если вам нужен бот с высокой нагрузкой или требующий распределенной архитектуры, Java может быть предпочтительным выбором.

Также является кроссплатформенным языком, что означает, что бот, разработанный на Java, может работать на разных операционных системах без изменений в коде. В Java есть компилятор JIT, который компилирует байт-код в отдельную машину, что позволяет напрямую вызывать скомпилированный код. Эта функция помогает увеличить скорость и эффективность языка. Наглядная схема выполнения любого приложения написанного на Java приведена на Рисунок 2.

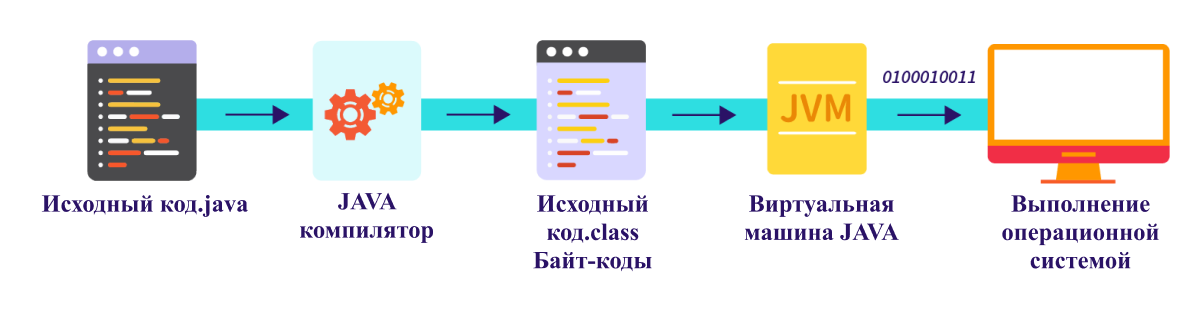


Рисунок – Порядок исполнения программ на java

Java имеет множество мощных фреймворков, таких как Spring и JavaFX, которые облегчают разработку приложений, включая Telegram-ботов. Эти фреймворки предоставляют различные инструменты и функциональность для разработки по-настоящему сложных приложений, например, Java EE предоставляет API для работы с базами данных, механизмы масштабирования и управления транзакциями. Кроме того, существуют библиотеки и инструменты, позволяющие интегрировать Java с различными сервисами, такими как SOAP и RESTful веб-службы, облачные платформы и многое.

Чаще Java-код быстрее чем Python, но читабельность его ниже и длина его, как правило, больше. Например, тест бинарного дерева может выполняться в Python в десять раз медленнее, чем в Java.

В Java также есть поддержка асинхронности. Начиная с версии Java 5, был добавлен пакет java.util.concurrent, который содержит классы и интерфейсы для работы с многопоточностью и асинхронностью. В Java 8 были добавлены функциональные интерфейсы CompletableFuture и CompletionStage, которые позволяют создавать асинхронный код. Кроме того, в Java есть библиотеки, такие как Netty и Vert.x, которые предоставляют механизмы для работы с асинхронными операциями.

3.1.3 PHP

PHP (Hypertext Processor) — это язык сценариев и интерпретатор общего назначения, который находится в свободном доступе и широко используется для веб-разработки. Этот язык используется в основном для сценариев на стороне сервера, хотя его также можно использовать для сценариев командной строки и, в ограниченной степени, для настольных приложений. Аббревиатура PHP первоначально была получена из Personal Home Page Tools, но теперь она расшифровывается как PHP: препроцессор гипертекста, который в документации PHP Group описывается как «рекурсивная аббревиатура».

При использовании для сценариев на стороне сервера PHP добавляется на веб-страницу с целью создания динамического содержимого при доступе к странице через клиентский браузер. Веб-сервер запускает сценарий перед передачей страницы в браузер. Для поддержки этого процесса веб-сервер требует, чтобы на сервере был установлен PHP вместе с синтаксическим анализатором PHP — либо синтаксическим анализатором Common Gateway Interface (CGI), либо серверным модулем [3].

Этот язык крайне популярен для веб-разработки, согласно статистике w3techs (Рисунок 3) PHP продолжает использоваться почти 80% всех веб-сайтов на некоторых основных платформах, таких как WordPress и Facebook [4].

PHP является кроссплатформенным языком, но его кроссплатформенность может быть ограничена зависимостью от конкретной операционной системы или веб-сервера. Например, если код PHP использует функции, которые доступны только в определенной операционной системе, то он не будет работать на других операционных системах. Также, если PHP-код использует расширения, которые не поддерживаются на определенной платформе, то он также не будет работать на этой платформе.

Но также согласно исследованию проведенному StackOverflow является одним из самым ужасных для выбора как основного языка 62,7% разработчиков, которые выбрали этот пожалели об этом и хотели бы сменить его [1].

Используя PHP, очень легко создать плохой код. Другие языки имеют гораздо больше ограничений. Мы также видим много проблем с безопасностью в PHP. И проблема здесь та же. На PHP очень легко сделать плохой и небезопасный код. Но PHP не является дырой в безопасности и не обречен на уродливый код, если вы кодируете правильно. Разработчики ненавидят PHP, потому что у вас больше шансов получить ошибки с языком, который дает так много свободы.

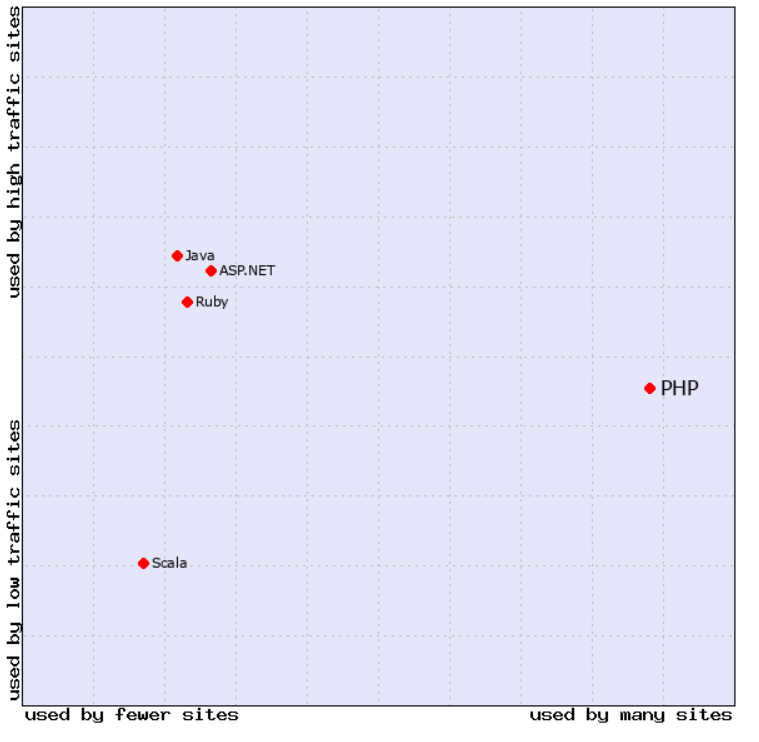


Рисунок - Частота использования PHP на рынке по трафику и применению по сравнению с другими языками.

Также большим недостатком PHP является его слабая типизация, что часто приводит к непредсказуемым ошибкам и во времени осложняет разработку, как крупных, так и малых проектов. Блокировка потоков выполнения, что делает невозможным или крайне затрудняет написание асинхронных функций даже с использованием сторонних библиотек.

3.1.4 Сравнение языков

Для сравнения, ниже представлена таблица рассмотренных выше языков по ряду основных критериев:

Таблица - Сравнительная таблица языков программирования

|  |  |  |  |
| --- | --- | --- | --- |
| Программная  платформа  Критерий | PHP | Python | JAVA |
| Кроссплатформенность | Частично | + | + |
| Типизация (Сильная/ слабая) | Слабая | Сильная | Сильная |
| Типизация  (Статичная/Динамичная) | Стат. | Динам. | Стат. |
| Универсальный | Только WEB | + | + |
| Целевые проекты | Малые/  средние | Малые/ средние | Крупные, масштабируемые |
| Поддержка асинхронности | Частично | + | + |

Исходя из таблицы 1 и изложенного выше языки Python и Java более универсальны и просты в использовании. В свою же очередь PHP ориентирован больше на разработку сайтов и серверных скриптов нежели хоть немного масштабируемой системы.

Также для разработки бота очень важно возможность асинхронного программирования, иначе блокировка потоков будет приводить к его постоянному зависанию.

Выбирая из Java и Python, был выбран Python, потому что:

* Это было пожеланием заказчика, т.к. уже имеются персонал и инфраструктура настроенная на этот язык
* Компания Oracle прекратила свою работу в России, что создает дополнительные риски, т.к. лицензия и документация Oracle не доступна уже сейчас
* Простота изучения и скорость разработки на Python выше
* Python хорошо подходит для данного проекта т.к. проект не планируется как высоконагруженный и масштабируемый, но при этом именно на Python проще всего будет его поддерживать

3.2 Основной Python фреймворк

Исходя из выбранного языка программирования, нужно определить какой фреймворк использовать для работы с Telegram Bot API, которая и будет основной при разработке данного бота. Для разработки Telegram-бота был выбран фреймворк AIOgram.

AIOgram — это фреймворк для разработки Telegram-ботов на языке программирования Python с использованием асинхронного программирования. Она построена на основе фреймворка asyncio и aiohttp, который позволяет создавать эффективные и отзывчивые асинхронные приложения [5].

Основные особенности и преимущества AIOgram:

1. Асинхронное программирование: используется преимущества асинхронной модели выполнения, что позволяет обрабатывать множество запросов одновременно и улучшает отзывчивость бота. Это особенно полезно для обработки параллельных задач, таких как обработка нескольких сообщений одновременно или взаимодействие с внешними сервисами, иначе пользователям бота при одновременном доступе к одной и той же задаче придется ждать приличное количество времени, пока она не будет завершена для каждого из них.
2. Простота использования: AIOgram имеет простой и интуитивно понятный интерфейс, который делает разработку Telegram-ботов более удобной. Она предлагает удобные методы и события для обработки входящих сообщений, обновлений и взаимодействия с Telegram API.
3. Обширная документация и активное сообщество: AIOgram имеет хорошо задокументированное API и обширную документацию, которая облегчает изучение и использование библиотеки. Кроме того, наличие активного сообщества разработчиков обеспечивает поддержку и обмен опытом при разработке Telegram-ботов.
4. Машина состояний: в AIOgram качественно произведена реализация машины состояний. Машина состояний AIOgram представляет собой граф, в котором вершинами являются состояния бота, а ребрами - переходы между ними. Каждое состояние определяет, какой тип сообщений оно может обрабатывать и какие действия должны быть выполнены при переходе в другое состояние.

AIOgram позволяет создавать несколько параллельных машин состояний для обработки различных типов сообщений, таких как текстовые сообщения, фотографии, аудио и т.д. Благодаря асинхронной модели программирования, бот может одновременно обрабатывать несколько запросов от пользователей и быстро отвечать на них. Машина состояний AIOgram предоставляет удобный способ организации логики бота и упрощает процесс разработки и поддержки Telegram-ботов на Python.

1. Удобные встроенные фильтры: AIOgram также предоставляет удобный способ фильтрации входящих сообщений и запросов от пользователей. Фильтры позволяют определять, какие сообщения должны быть обработаны ботом, а какие игнорированы.

Например, можно создать фильтр для обработки только текстовых сообщений, содержащих определенные ключевые слова или фразы. Такой фильтр может быть полезен для автоматической обработки запросов от пользователей, связанных с определенной темой или категорией, например определение администратора.

Фильтры в AIOgram могут быть созданы с помощью декораторов и легко настраиваются для различных типов сообщений и запросов. Это позволяет боту быстро и эффективно обрабатывать большой объем входящих данных и отвечать на запросы пользователей в режиме реального времени. В большинстве других фреймворков для этого отдельного механизма не предусмотрено, и разработчик вынужден самостоятельно прописывать длинные лямбда-функции фильтрации.

3.3 СУБД

В соответствие с одним из ключевых требований при разработке – наличия базы данных, в качестве СУБД была выбрана PostgreSQL.

PostgreSQL — это мощная и популярная объектно-реляционная система управления базами данных (СУБД), которая предлагает множество преимуществ и характеристик, делающих её привлекательным выбором для многих проектов [6].

Далее приведено детальное обоснование использования PostgreSQL в качестве СУБД:

1. Надежность и стабильность: PostgreSQL известен своей высокой надежностью и стабильностью. Он прошел множество тестов временем и активно разрабатывается, и поддерживается сообществом разработчиков. Благодаря своей надежности, PostgreSQL является основной базой данных для множества крупных проектов, включая компании, учреждения и государственные организации.
2. Расширяемость и гибкость: PostgreSQL предлагает широкий набор функциональных возможностей и поддерживает множество расширений. Он поддерживает языки программирования, такие как Python, Perl, C/C++, Java, что позволяет разработчикам создавать расширения, функции и процедуры на своих предпочитаемых языках. Богатая функциональность PostgreSQL позволяет адаптировать СУБД под разнообразные требования проекта.
3. Поддержка стандартов: PostgreSQL активно поддерживает стандарты ANSI SQL и SQL:2008. Это означает, что код, разработанный для PostgreSQL, будет более переносимым между различными СУБД, что упрощает перенос проектов и повышает их совместимость. PostgreSQL также поддерживает множество расширений стандарта SQL, таких как JSON, XML, географические данные и многое другое.
4. Масштабируемость: PostgreSQL обладает высокой масштабируемостью и способен обрабатывать большие объемы данных. Он поддерживает горизонтальное масштабирование с использованием репликации и разделения данных, что позволяет распределить нагрузку и улучшить производительность при необходимости. PostgreSQL также предлагает возможности партиционирования для эффективного управления большими таблицами.
5. Богатая функциональность: PostgreSQL предлагает множество расширенных возможностей, которые делают его привлекательным для различных проектов. Это включает в себя поддержку геоданных и пространственных запросов, полнотекстовый поиск, транзакционную поддержку с высокой согласованностью данных, встроенную поддержку JSON и XML, возможность создания собственных типов данных и многое другое. Это позволяет разработчикам реализовывать сложные функциональные требования без необходимости использования сторонних расширений.
6. Высокая производительность: PostgreSQL известен своей высокой производительностью. Он обладает оптимизированным движком запросов, предоставляет индексы для ускорения поиска и фильтрации данных, а также предлагает возможность оптимизации запросов с помощью анализатора запросов. PostgreSQL также поддерживает параллельную обработку запросов и предоставляет механизмы кэширования для повышения производительности.
7. Бесплатная и открытая лицензия: PostgreSQL распространяется под лицензией PostgreSQL, которая является бесплатной и открытой. Это означает, что вы можете использовать PostgreSQL без необходимости покупать коммерческую лицензию, что делает его доступным для широкого круга проектов и организаций.
8. Активное сообщество и поддержка: PostgreSQL имеет активное и преданное сообщество разработчиков, которые обеспечивают поддержку, развитие и полноценную русскоязычную документацию. Существуют форумы, рассылки, онлайн-ресурсы и множество сторонних библиотек и инструментов, связанных с PostgreSQL. Это обеспечивает доступ к знаниям, советам и решениям проблем при работе с PostgreSQL.

В совокупности всего вышесказанного, PostgreSQL предлагает множество преимуществ и характеристик, которые делают его привлекательным выбором для различных проектов. Он обладает надежностью, масштабируемостью, высокой производительностью, богатой функциональностью и активной поддержкой сообщества, поэтому была выбрана именно данная СУБД.

4 АРХИТЕКТУРА ПРИЛОЖЕНИЯ

4.1 Общая архитектура приложения

Архитектура Telegram-бота играет важную роль в его эффективной работе и взаимодействии с пользователями. Она должна быть гибкой, масштабируемой и удобной для разработки и поддержки. Ниже рассмотрим основные принципы и компоненты архитектуры Telegram-бота.

Для проектирования программного обеспечения часто прибегают к различным архитектурным шаблонам – общее решение для типичных проблем проектирования, которое может быть использовано для создания архитектуры программного обеспечения. Шаблон определяет структуру компонентов системы, их взаимодействие и зоны ответственности, что позволяет упростить процесс разработки и поддержки приложения. Для разработки и поддержки достаточно малого веб-продукта эффективнее всего будет использовать архитектурный шаблон Model-View-Controller (MVC) (Рисунок 4) в данном случае:

1. **Модель (Model)**: Модель представляет собой модуль данных бота. Она содержит данные и методы, которые через контролер обрабатывают запросы от пользователя и выполняют соответствующие действия с БД.
2. **Представление (View)**: Представление отвечает за отображение данных пользователю. В случае с Telegram-ботом, представление может быть текстовым сообщением, изображением, кнопками и другими элементами пользовательского интерфейса. Представление получает данные от модели и генерирует соответствующий ответ для отправки пользователю через Telegram API.
3. **Контроллер (Controller)**: Контроллер служит связующим звеном между моделью и представлением. Он обрабатывает входящие запросы от пользователя, интерпретирует их и вызывает соответствующие методы модели для выполнения действий. Затем контроллер получает результат от модели и передает его представлению для формирования ответа пользователю. Контроллер также может обрабатывать команды, кнопки и другие элементы пользовательского ввода. В нашем случае контролером будет являться обработчики сообщений для разных видов ролей и ситуаций (handler)

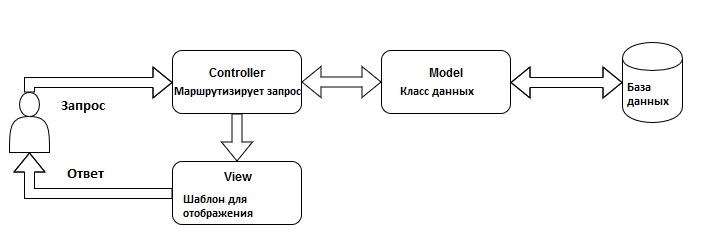


Рисунок – Схематическое представление шаблона MVC

Таким образом, используя шаблон проектирования MVC, появляется возможность разделить логику Telegram-бота на модель, представление и контроллер. Это поможет создать более структурированный код, облегчить его поддержку и расширение, а также лучше разделить ответственности между компонентами приложения.

4.2 Режим работы бота

Бот может работать в двух режимах:

1. Как автономное приложение (long polling) – постоянно запущенный скрипт, который на регулярной основе опрашивает сервера Telegram. После чего в соответствии с алгоритмами внутри самого приложения выполняет различные функции.
2. WebHook - это механизм, который позволяет боту получать и обрабатывать входящие сообщения и запросы от пользователей в режиме реального времени. С помощью WebHook, бот может получать информацию о новых сообщениях, обновлениях чата и т.п., но ключевой минус использования бота на WebHook заключается в том, что нужен выделенный веб-сервер, который должен принимать запросы только по протоколу https, что в свою очередь требует валидный SSL-сертификат, который не бесплатен и иногда дорог в содержании.

Так как приоритет заказчика в таких моментах заключается в наименьших затратах на обслуживание и содержание, поэтому был выбран режим работы, основанный на периодическом опросе сервера.

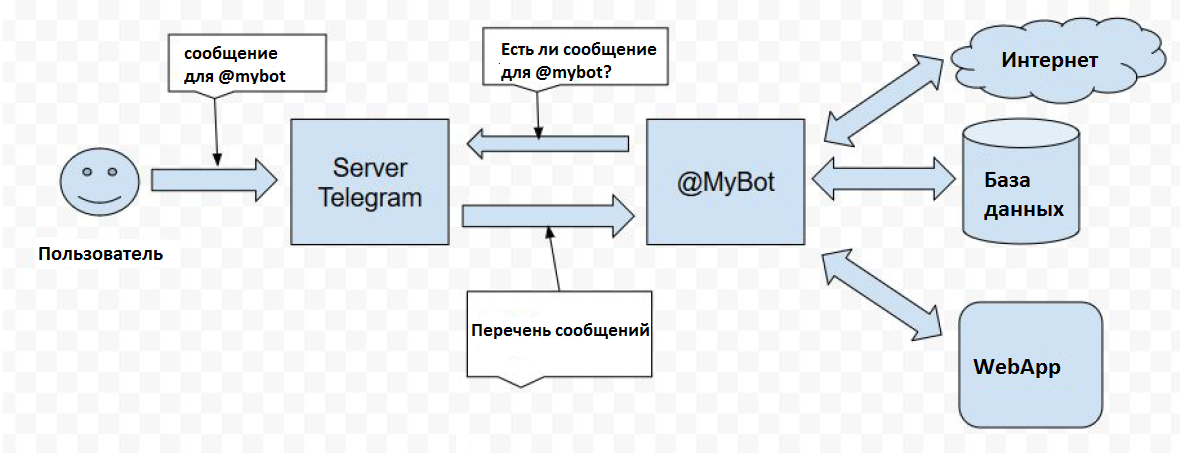


Рисунок - Диаграмма работы Telegram-Бота для автономного приложения

5 СЦЕНАРИИ ВЗАИМОДЕЙСТВИЯ С СИСТЕМОЙ

Сценарии взаимодействия с системой будет описываться диаграммой вариантов использования — это графическое представление сценариев использования системы или её компонентов. Она отображает взаимодействие между пользователями и системой, а также между различными компонентами системы. Диаграмма вариантов использования может быть использована для описания функциональных требований системы, а также для анализа и оптимизации её работы.

Диаграмма вариантов использования состоит из следующих элементов:

1. Актеры — это пользователи или другие системы, которые взаимодействуют с системой.

2. Варианты использования — это сценарии использования системы или её компонентов. Каждый вариант использования описывает определенную функцию или действие, которое может быть выполнено пользователем или системой.

3. Связи — это отношения между актерами и вариантами использования. Они показывают, какие актеры могут выполнять какие варианты использования.

Диаграмма вариантов использования может быть представлена в виде прямоугольников, соединенных линиями. Каждый прямоугольник представляет один вариант использования, а линии показывают связи между актерами и вариантами использования.

Ниже будут представлены схемы взаимодействия пользователя и бота для каждой роли:

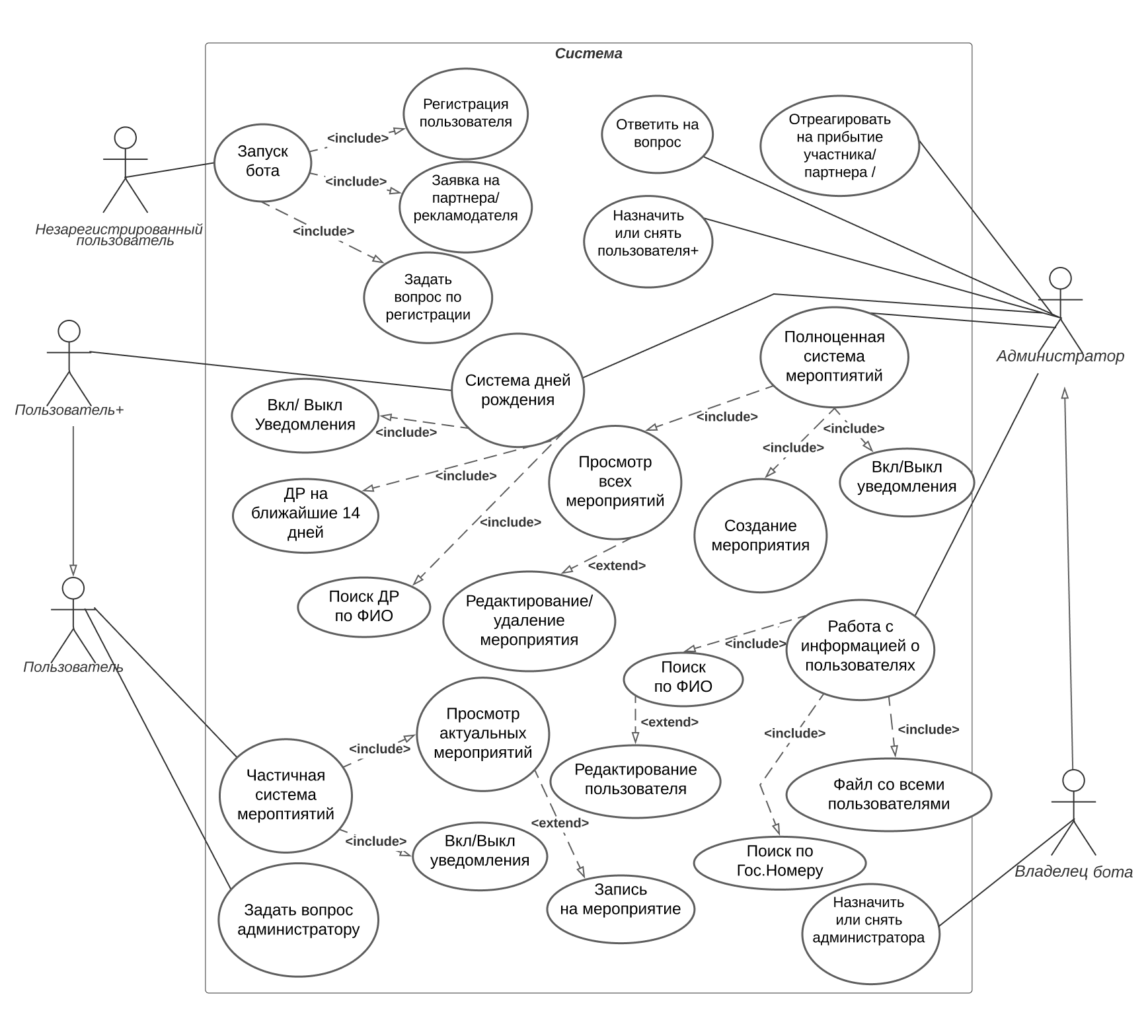
****

Рисунок – Диаграмма вариантов использования

6 РАЗРАБОТКА БОТА

6.1 Структура бота

Перед началом работы необходимо импортировать все нужные библиотеки с помощью pip, это:

* Psycopg2 – необходима для прямой работы с БД
* AIOgram – основная библиотека проекта
* APScheduler – нужен для реализации планировщика задач в асинхронном режиме, для формирования оповещений.
* OpenPyXL – нужен для формирование структурированных отчетов формата excel
* AIOgram-calendar – обращение к календарному алгоритму

Перед реализацией структуры приложения, соответствующей заданной архитектуре - MVС шаблону, следует подготовить созданный проект. С начала необходимо разделить проект на разные пакеты, которые бы соответствовали архитектурной структуре приложения на MVC, но со спецификой Telegram-бота (Рисунок 7), а именно:

* Database – пакет, где будет происходить основное взаимодействие с базой данных (Model)
* Handlers – пакет, где будут находится основные обработчики пользовательского ввода (Controller)
* Keyboards – пакет отвечающий за отображение у пользователя клавиатур и кнопок, т.е. графического интерфейса (View)
* Вспомогательные пакеты:
  + Misc – пакет, отвечающий за дополнительные функции такие как, анти-мат, форматирование текста, набор состояний FSM и предобработка сообщений
  + Filter – пакет, отвечающий за фильтры, которые должны отличать разные роли пользователей
  + Res – статические ресурсы такие как картинки, строки реплик бота и т.п.

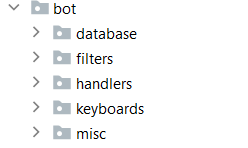


Рисунок – Структура проекта

Запуск бота должен осуществляться с регистрации всех написанных объектов, а именно: фильтров, команд, обработчиков, соединения с базой данных.

Данные процессы происходят в сопрограмме \_\_on\_start\_up, которая в свою очередь вызывает функции и сопрограммы инициализации всех объектов и запуска планировщика. Эта функция автоматически вызывается фреймворком AIOgram при указании её в качестве параметра on\_startup функции диспетчера start\_polling (Рисунок 8).

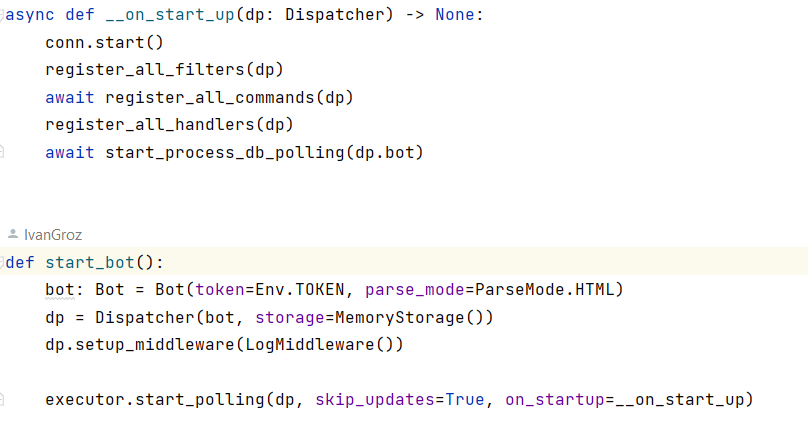


Рисунок – Фрагмент скрипта main.py

Каждая сопрограмма или функция, где происходит регистрация, находится каждая в своем пакете в скрипте main, куда импортируются скрипты, где регистрируются соответствующие пакету объекты (Рисунок 9 -11).

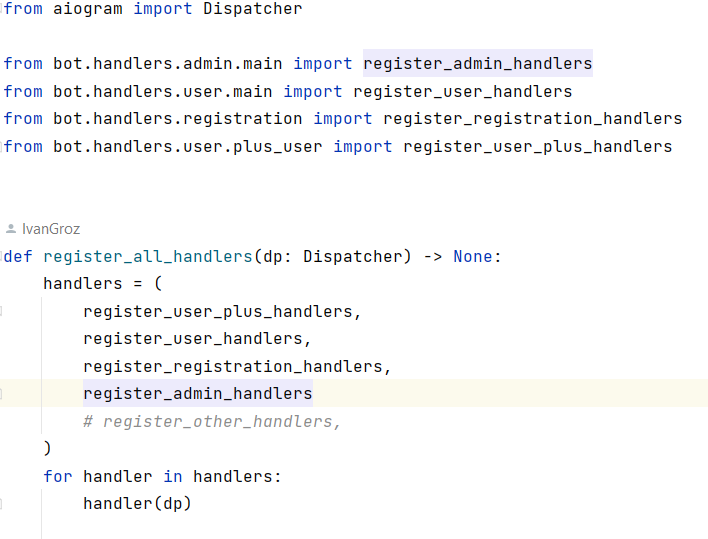


Рисунок – Регистрация всех обработчиков в скрипте handlers\main.py



Рисунок – Регистрация всех фильтров в скрипте filters\main.py

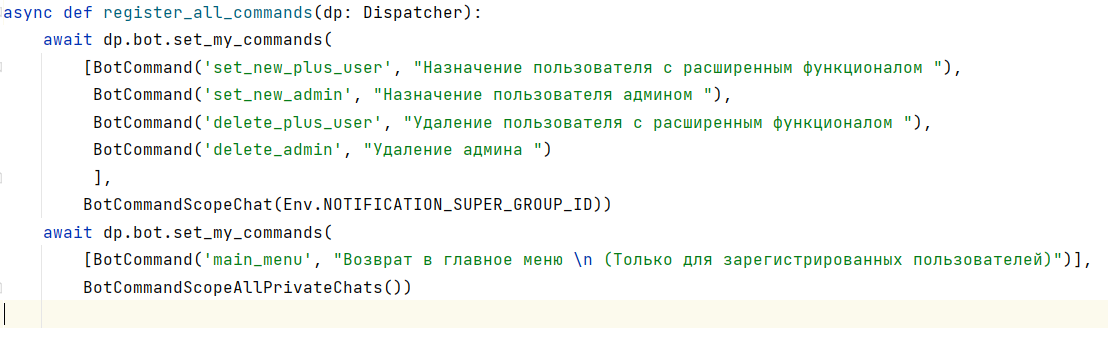


Рисунок – Регистрация всех команд в скрипте commands.py

Тут же в скрипте commands.py (Рисунок 11) задается, из переменных окружения Env, область видимости команд, а именно чтобы команды назначения и снятия ролей были доступны только в специальной группе, уполномоченными пользователями.

Обработчики событий в свою очередь тоже разделены на пакеты, каждый соотносящийся с определенной ролью в системе (Рисунок 12).

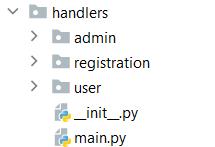


Рисунок – Структура обработчиков событий

Условно пакеты обработчиков разделены на роли:

* Управляющая – это администратор, владелец и обработчик действий в специальной группе
* Пользовательская — это обычный пользователь и с расширенной функциональностью
* Гость – это незарегистрированный пользователь

Уже внутри этих обработчиков вызываются сопрограммы, которые в свою очередь обращаются к View составляющей бота, через пакет keyboards.

6.2 База данных

Структура базы данных крайне проста и состоит из 4х основных таблиц и 6 вспомогательных функций (Рисунок 13-14):

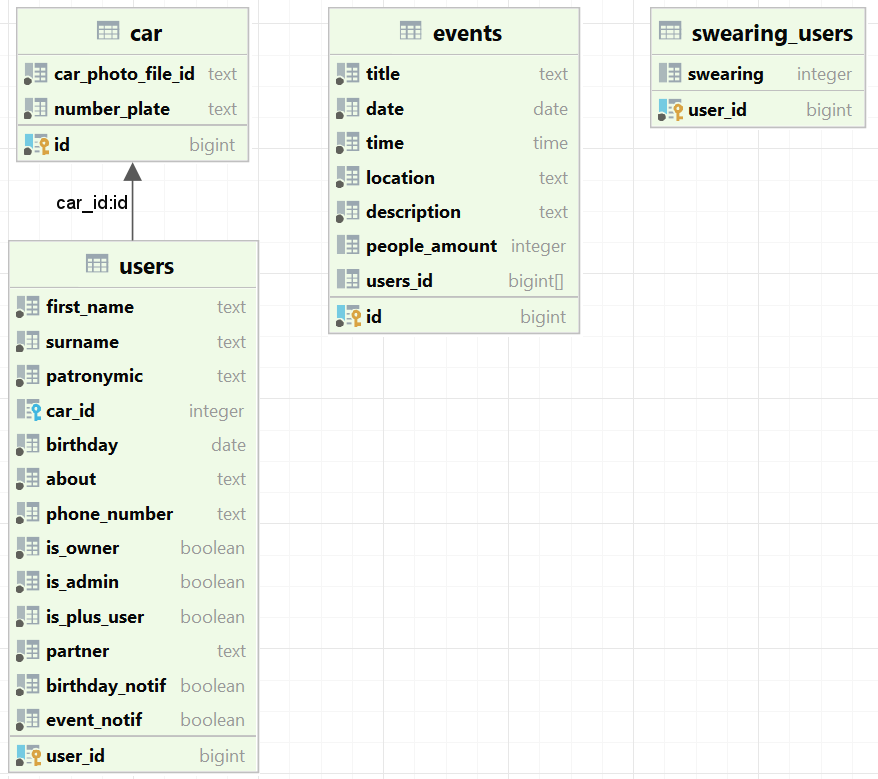


Рисунок – Структура таблиц базы данных

Назначение таблиц базы данных:

* Users – хранение данных о пользователе
* Car – хранение данных об автомобиле пользователя
* Events – таблица, куда добавляются и хранятся новые события и актуальная информация о подписавшихся на событие участниках
* Swearing\_users – таблица, суть которой хранить id тех пользователей, которые использовали ненормативную лексику в группе.

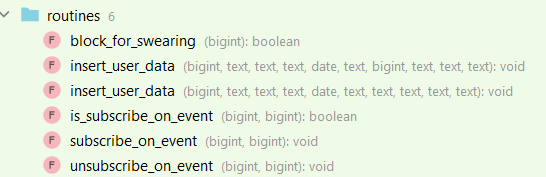


Рисунок – Вспомогательные функции базы данных

За инициализацию и соединение с базой данных отвечает функция start в скрипте database.py, работающий на библиотеке psycopg2, внутри которой находится класс connection, отвечающий за соединение с БД. Объект этого класса является статической переменной, к которой обращаются функции для доступа к запросам в БД. В этом же скрипте основные методы инициализации и доступа к БД (Рисунок 15), пример доступа к БД через connection на Рисунке 16:



Рисунок – Функция инициализации подключения к БД в скрипте database\database.py

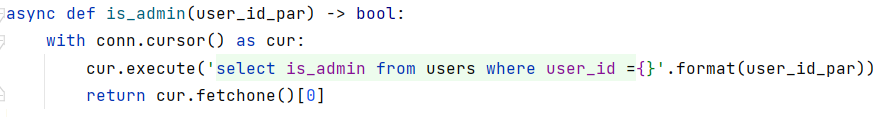


Рисунок – Доступ к БД через connection для получения значения переменной таблицы users, is\_admin

Также в пакете, отвечающем за обращение к БД реализован планировщик задач, который взаимодействует с информацией из базы данных один раз в сутки и раз в неделю. Планировщик вызывает сопрограммы получения из базы данных ДР пользователей, на текущий день, и мероприятия клуба, которые должны произойти на следующий день и раз в неделю обнулить штрафы людей, которые должны быть наказаны за употребление ненормативной лексики (Рисунок 17).

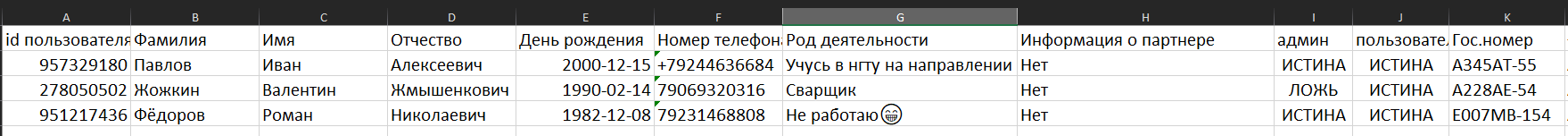


Рисунок – Сопрограмма планировщика, которая опрашивает БД из скрипта database\database\_polling.py

Решение задачи формирования excel-файла с полной информацией о пользователях также является частью пакета database (Рисунок 18), благодаря библиотеки openpyxl удалось, сформировать удобно структурированный excel-файл (Рисунок 19).



Рисунок – Сопрограмма составляющая excel файл с информацией о всех пользователях в скрипте database\import\_in\_file.py



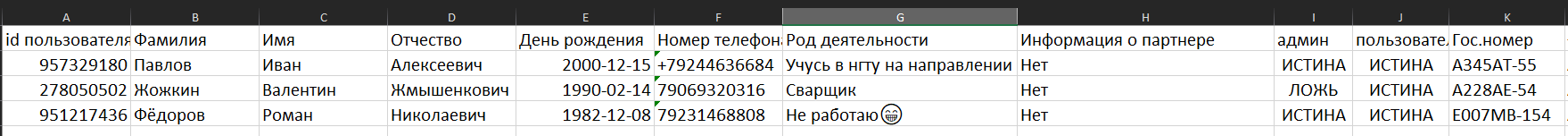


Рисунок – Результат формирования excel-файла

6.3 Визуальная часть бота

Визуальная часть бота располагается в пакете keyboards, основная часть всех интерфейсов, а именно встроенные в сообщение клавиатуры и клавиатурные кнопки, это inline.py (Рисунок 20), reply.py (Рисунок 21) соответственно.

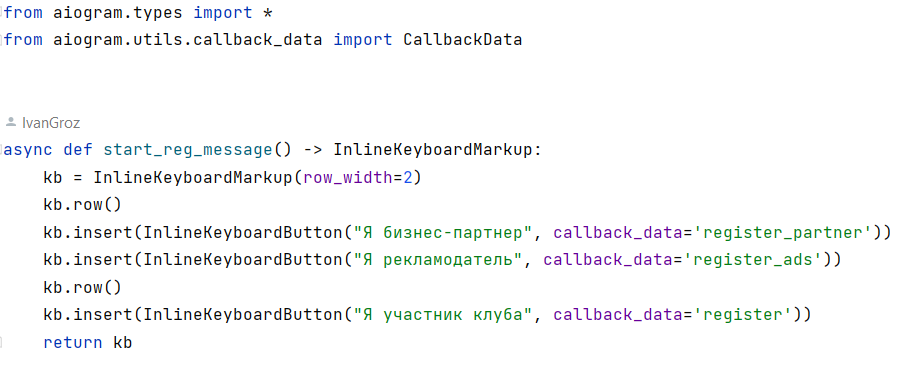


Рисунок – Пример сопрограммы реализующая встроенную клавиатуру на приветственном сообщении bot/keyboards/inline.py



Рисунок - Пример сопрограмм возвращающие клавиатурные кнопки в скрипте bot/keyboards/reply.py

Специализированные встроенные клавиатуры реализовывают замкнутые графические системы в данном боте, это:

* Встроенная клавиатура выбора даты рождения - date\_picker
* Встроенная клавиатура выбора календарного дня - simple\_calendar
* Встроенная клавиатура слайдер именинников - birthday\_slider
* Встроенная клавиатура слайдер мероприятий - events\_slider
* Встроенная клавиатура администраторский слайдер мероприятий - events\_slider\_admin

Суть реализации этих графических систем в том, что инициализируется набор CallbackData, где описываются данные и действия. Инициализируется начальная встроенная клавиатура, где в каждый пункт CallbackData добавляется название, того действия за которое отвечает кнопка и данные, которые должны изменяться в случае нажатии кнопки. Внутри класса реализована сопрограмма, которая при нажатии кнопки вызывается, внутри проверяется состояние каждого элемента CallbackData и в зависимости от показателя действия пересоздается новая встроенная клавиатура и в случае получения конечного результата возражает True и выбранные пользователем данные, если нет, то графическая система продолжает существовать.

6.4 Дополнительные функции

Система анти-мата, данная система реализована на основе прямого сравнение предварительно переведенный в нижний регистр строки с объёмной коллекцией нецензурных слов и в случае совпадения все символы нуждающиеся в экранировании по формату MarkdownV2 экранируются, сообщение пользователя в группе удаляется, выносится предупреждение и бот цензурирует сообщение пользователя и дублирует его и в случае превышение лимита пользователь ограничивается в группе и не может писать сообщения на протяжении недели (Рисунок 22).

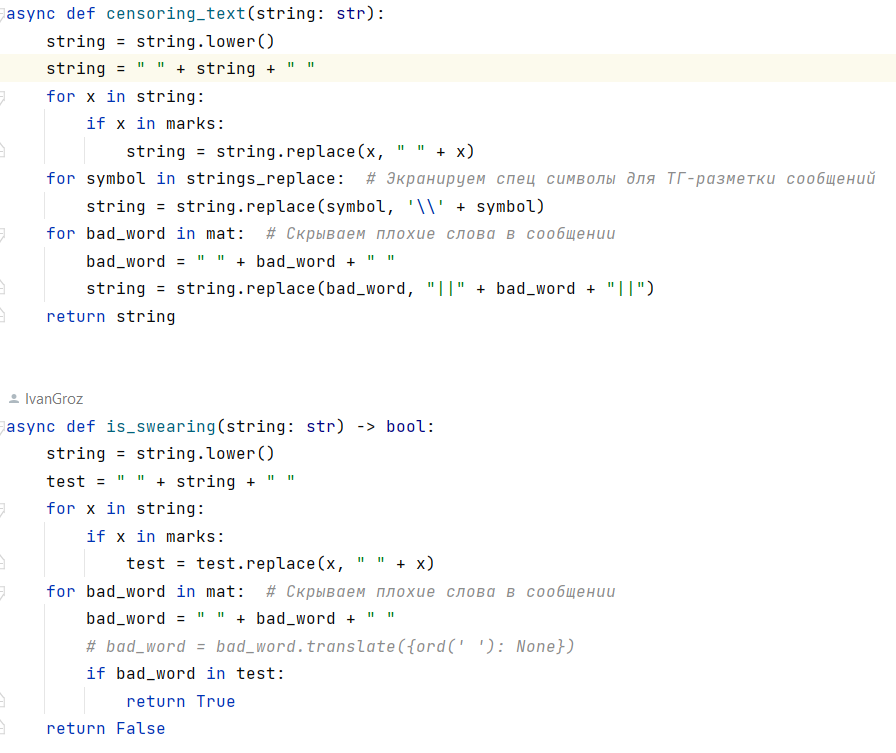


Рисунок – Сопрограммы, реализующие логику определения нецензурной лексики и форматирование строки для формата MarkdownV2 в скрипте misc/anti\_swearing.py

7 ТЕСТИРОВАНИЕ И АПРОБАЦИЯ

7.1 Ручное тестирование

Ручное тестирование – этап контроля качества, при котором инициируется процесс проверки функциональности, надежности, удобства использования и других аспектов программного продукта путем ручного взаимодействия с ним, то есть проверка без использования сторонних программных инструментов. В этом процессе тестировщик выполняет заданные тестовые сценарии, вводит данные в программу, проверяет её реакцию на различные входные данные и ситуации, а также выявляет и документирует ошибки и недостатки.

Далее будет приведены результаты ручного тестирование Telegram-бота, тестирование будет проводиться по ролям в частном чате и в специальной группе.

7.1.1 Незарегистрированный пользователь

Предполагается, что запуск бота будет осуществляться при переходе по ссылке на бота или при поиске его имени «@BMWClubNSK\_bot» в Telegram. При переходе по ссылке в чат с ботом, пользователь видит сообщение с описанием бота и кнопку «Запустить» (Рисунок 23).

После нажатия на кнопку запуска бота, пользователь автоматически отправляет команду /start, бот присылает приветствие, поясняющее то, что требуется от пользователя во время регистрации, ссылкой на администратора клуба и со встроенной клавиатурой, имеющие кнопки: «Я бизнес-партнер», «Я рекламодатель», «Я участник клуба». (Рисунок 24)

Для начала регистрации пользователя в системе, ему необходимо выбрать кнопку «Я участник клуба», после чего приветственное сообщение удалится и бот отправляет предупреждение о том, что в случае ошибки при вводе данных, форму регистрации придется заполнять заново. Запрашивает ФИО и у пользователя появляется клавиатура с пунктами: «Получить помощь по регистрации», «Допустил ошибку/ Начать заново», которые можно нажать на любом этапе регистрации. (Рисунок 25)

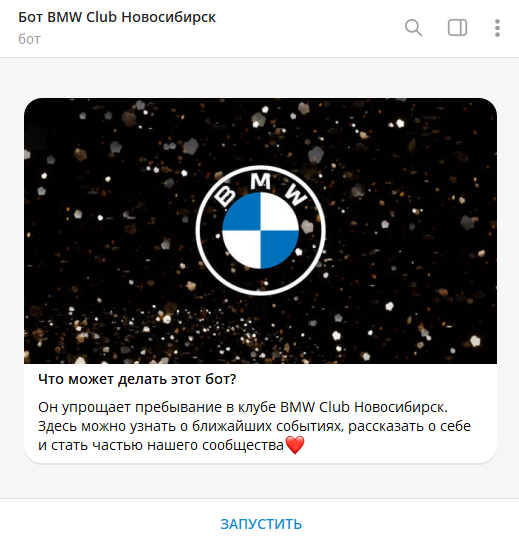


Рисунок – Начальный экран при переходе в чат с ботом

После ввода ФИО, пользователь указывает дату рождения в специальной встроенной клавиатуре, которая динамически изменяется, выбор даты разделен на этапы: выбора года, выбора месяца, выбора дня. После конечного этапа выбора дня, сообщение со встроенной клавиатурой удаляется и бот присылает результат выбора в формате YY-MM-DD (Рисунок 26).

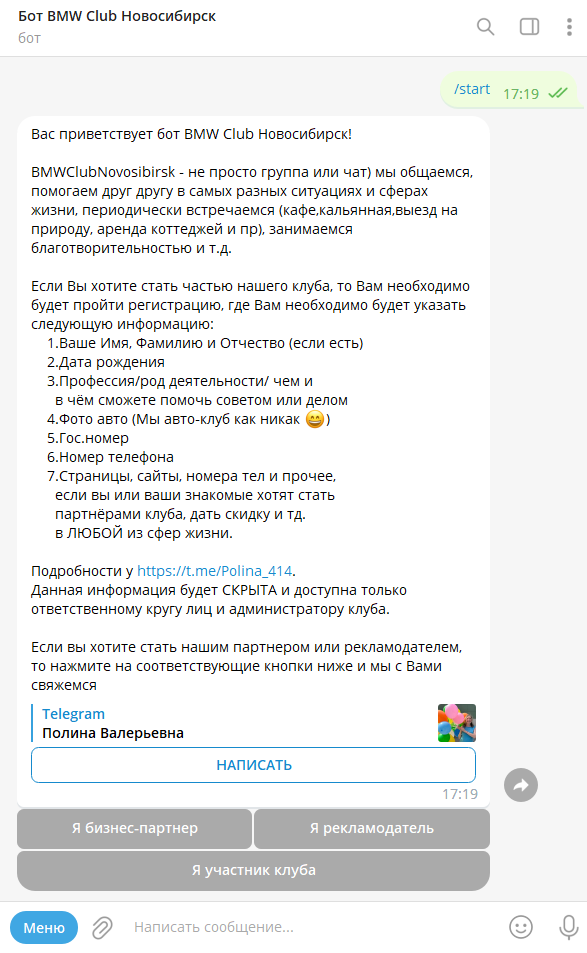


Рисунок – Экран при запуске бота командой /start

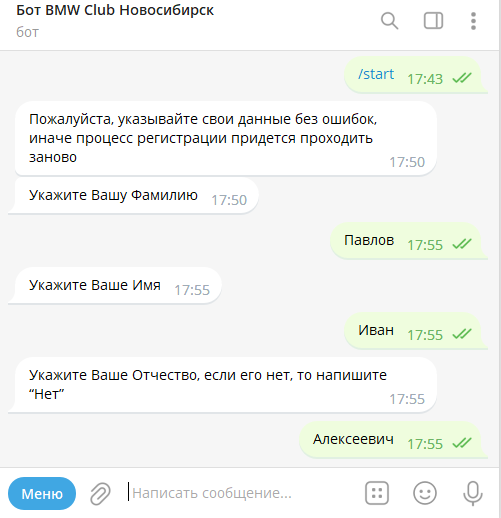


Рисунок – Начало регистрации участника клуба

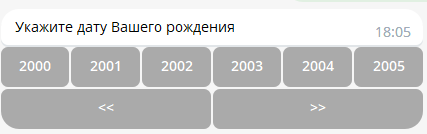
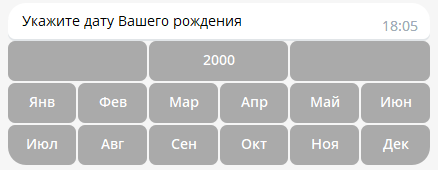
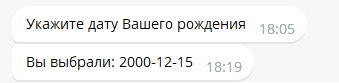


Рисунок - Этапы ввода информации о дне рождения

После ввода дня рождения незарегистрированный пользователь отправляет информацию о роде деятельности и отправляет фото автомобиля, после чего бот отправляет сообщение с запросом ввода гос. номера авто с примером заполнения и встроенной клавиатурой с кнопкой «Мой гос. номер другого формата», нужная для тех людей у кого автомобиль зарегистрирован не в России или зарегистрированы спец. номера, для них проверка регулярным выражением на ввод корректного формата не учитывается. (Рисунок 27)

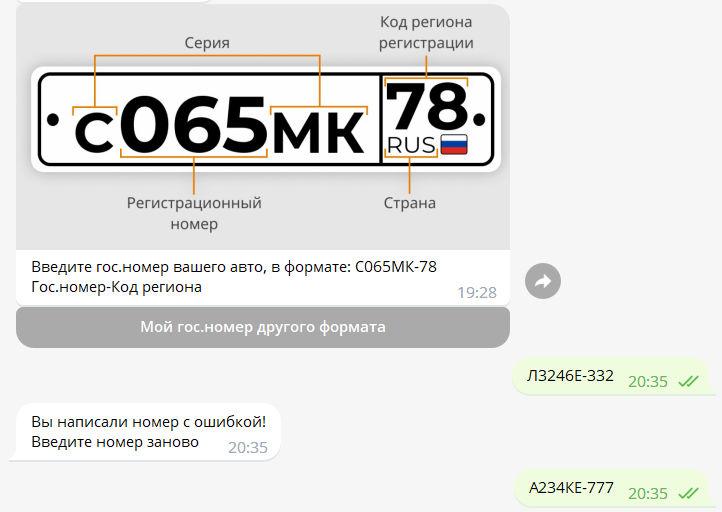
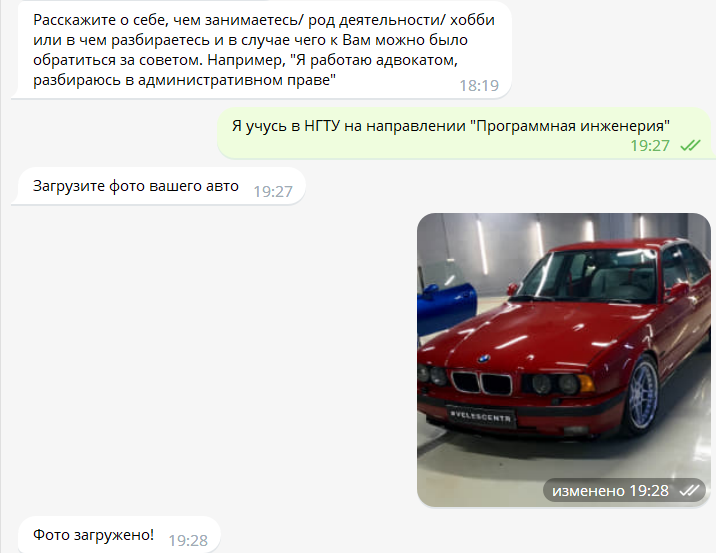


Рисунок – Ввод информации о роде деятельности, загрузка фото авто и ввод гос. номера автомобиля

Далее бот запрашивает номер телефона и появляется клавиатурная кнопка «Отправить номер телефона», после её нажатия пользователь присылает боту свой номер телефона в виде контакта, за этим следует сообщение с вопросом о том, хотел бы участник клуба быть партнером или нет, и прикрепляются соответствующие кнопки встроенной клавиатуры, если нажать «Нет», то можно будет сразу завершить регистрацию нажав на кнопку встроенной клавиатуры, предварительно проверив введенные данные (Рисунок 28).

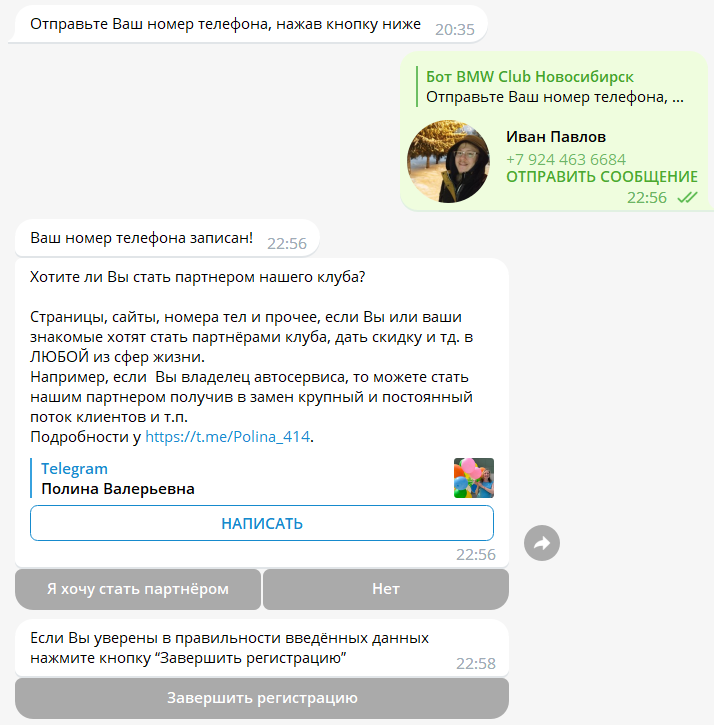


Рисунок – Отправка номера телефона, отказ от партнерства и предупреждение о завершении регистрации

После завершения регистрации бот отправляет сообщение об успешной регистрации и уже у зарегистрированного пользователя появляется клавиатура соответствующей функциональности (Рисунок 29), а также в специальную группу администраторов, в выделенной под этот тип уведомлений тему, приходит уведомление о новом участнике с ссылкой на зарегистрировавшегося пользователя .

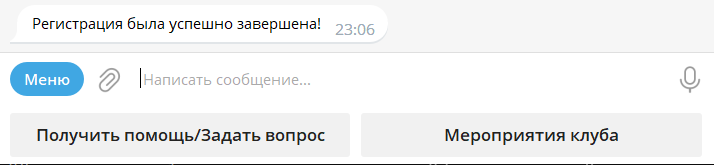


Рисунок – Успешная регистрация

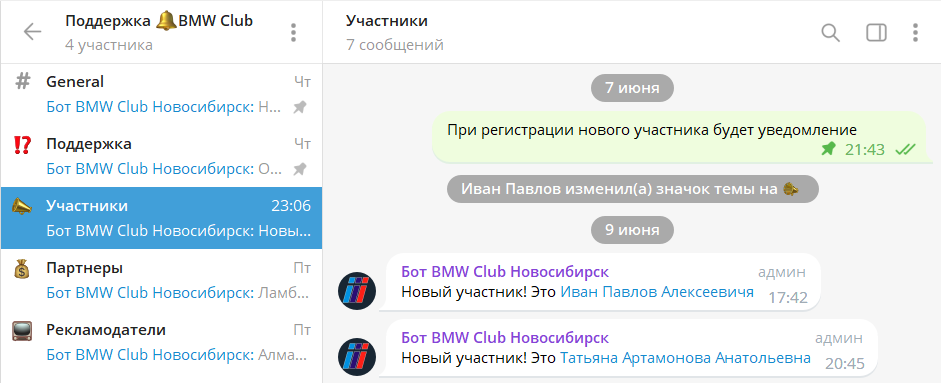


Рисунок – Уведомление о новом участнике

Протестируем работу функции поддержки пользователя, эта функция одинаково работает как для зарегистрированного пользователя, так и для незарегистрированного. При нажатии, в любой момент регистрации, клавиатурной кнопки «Получить помощь по регистрации», бот опрашивает пользователя о том, с чем у него возникли трудности и предлагает начать регистрацию заново (Рисунок 31).

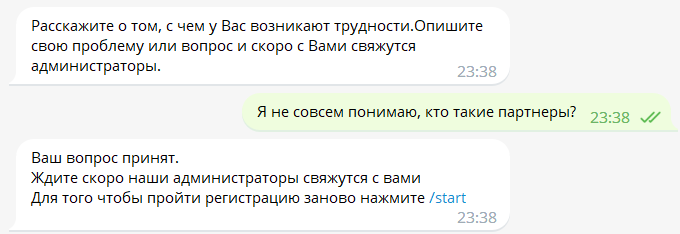


Рисунок – Получение помощи по регистрации

После данное обращение посылается в специальную группу для администраторов, в тему «Поддержка», со встроенной клавиатурой с двумя кнопками, «Ответить на вопрос» и «Отвечен!»

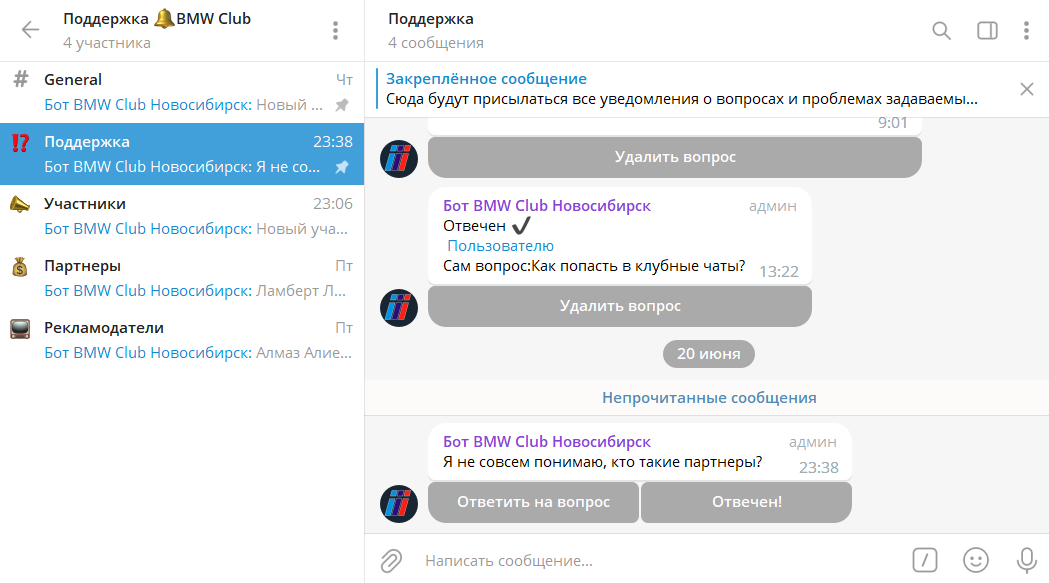
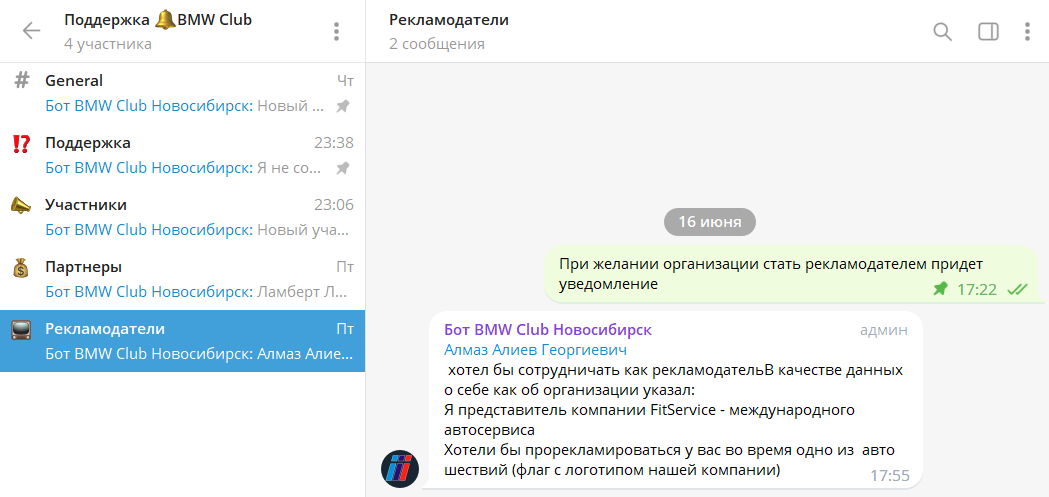
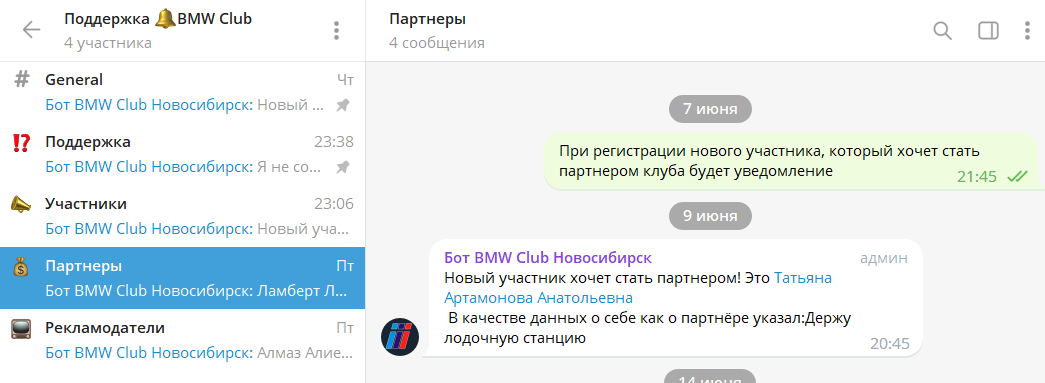


Рисунок – Уведомление о новом вопросе от пользователя

По аналогии с уведомлениями о новом участнике в специальную группу присылаются уведомления о партнерах/рекламодателях в соответствующие темы, обращения которых, могут быть оформлены в самом начале или конце регистрации, где им не нужно указывать дополнительную информацию, кроме ФИО и самого обращения. (Рисунок 33)

Рисунок -Уведомления о новых партнерах и рекламодателях



7.1.2 Участник клуба (обычный пользователь)

После того как незарегистрированный пользователь стал участником клуба, у него появляется клавиатура с системой актуальных мероприятий и поддержкой, которая работает абсолютно аналогично той, что есть у незарегистрированного пользователя (Рисунок 29). Протестируем систему актуальных клубных мероприятий. Нажимая на кнопку «Мероприятия клуба» открывается подменю с кнопками: «Ближайшие мероприятия» и «Отключить уведомления», по умолчанию у всех уведомления включены (Рисунок 34). Нажимая на «Отключить уведомления», кнопка меняется на «Включить уведомления», и пользователь отписывается от уведомлений о завтрашних мероприятиях (Рисунок 35).

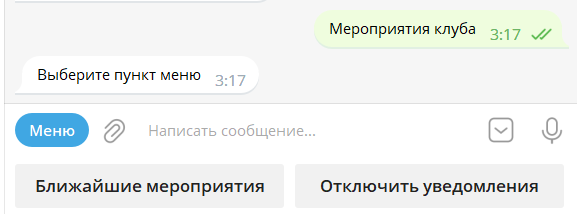


Рисунок – Пункт мероприятия клуба у обычного пользователя

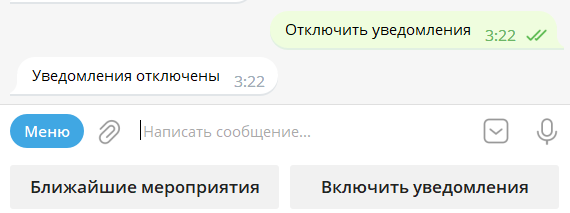
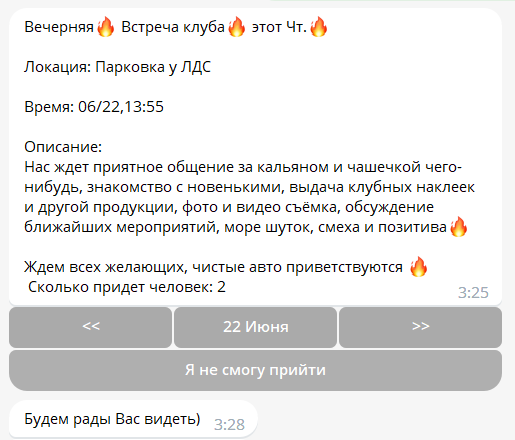
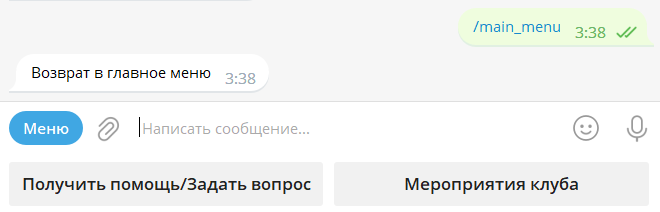


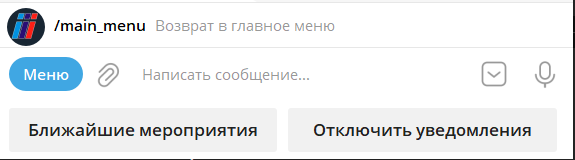
Рисунок – Отключение уведомлений о мероприятиях

Нажимая на «Ближайшие мероприятия» пользователю, отправляется графический элемент - слайдер мероприятий с датой просматриваемого мероприятия, в котором отображены ближайшие актуальные мероприятия, то есть те, которые ещё не прошли и кнопкой «Я приду/Я не смогу прийти», при нажатии на кнопку «Я приду», пользователь подпишется на мероприятие и количество человек, которое придет на мероприятие увеличится на 1 и наоборот (Рисунок 36).

Возврат во всех ролях в главное меню реализовано через единую команду /main\_menu протестируем её работу и продемонстрируем результат до и после её исполнения. (Рисунок 37).



Рисунок – Слайдер актуальных мероприятий

Рисунок – Результат до выполнения (выше) команды /main\_menu и после

Так же пользователям всех ролей приходит рассылка важных сообщений, и чтобы не потерять важное сообщение, по мере взаимодействия с ботом, оно закрепляется в чате (Рисунок 38).



Рисунок – Пример рассылки важного сообщения пользователю

Проверим как работает планировщик событий и дождемся, когда он сработает, обычному пользователю должен приходить только один тип уведомлений – о мероприятиях, причем в случае, если организатором запланировано несколько мероприятий в один и тот же день, в уведомлении о мероприятии будет упоминание о всех мероприятиях (Рисунок 39).

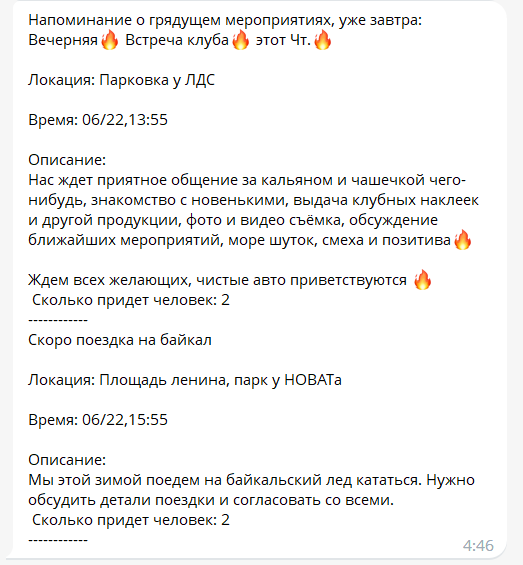


Рисунок – Уведомление о завтрашних мероприятиях

7.1.3 Пользователь+

Отличие данной роли от обычного пользователя в том, что у него есть доступ к системе дней рождения и дополнительных функций, которые должны будут появляться по мере внедрения и поддержки бота. Его главное меню имеет три пункта: «Мероприятия клуба», «Дни рождения» и «Получить помощь/ задать вопрос», данное меню обычный пользователь получает только после повышения его в пользователя+ в специальной группе администраторов, в теме «Главная», после чего бот присылает пользователю уведомление о его повышении и меняет главное меню (Рисунок 40).

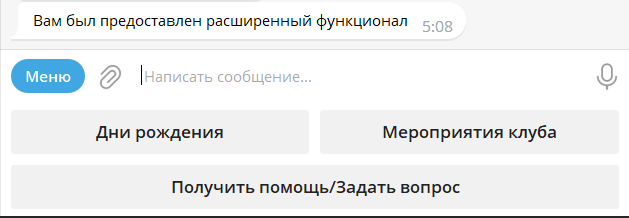


Рисунок – Получение роли пользователя+

При выборе пункта «Дни рождения», появляется клавиатура с кнопками: «Показать ДР на ближайшие 14 дней», «Отключить уведомления о ДР», «Узнать день рождения по ФИО». Протестируем работу каждого пункта меню.

При нажатии на кнопку «Отключить уведомления о ДР», кнопка меняется на «Включить уведомления о ДР», и пользователь отписывается от уведомлений о сегодняшних днях рождения (Рисунок 41).

При выборе пункта «Показать ДР на ближайшие 14 дней», бот присылает пользователю+ слайдер с информацией о пользователях отсортированными по дате рождения (Рисунок 42).

При выборе «Узнать день рождения по ФИО», бот присылает инструкцию о том как правильно вводить ФИО, чтобы поиск работал корректно, после ввода полного ФИО или его части, бот присылает список найденных по запросу пользователей и чтобы узнать день рождения нужного пользователя нужно нажать на ссылку под ним (Рисунок 43).

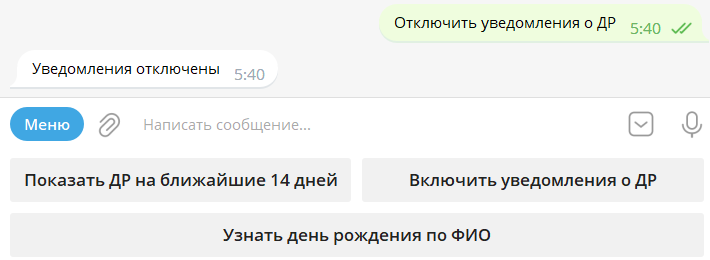


Рисунок – Отключение уведомлений о ДР

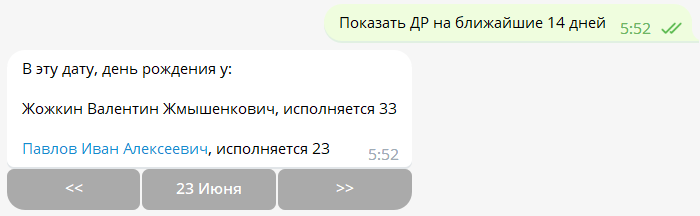
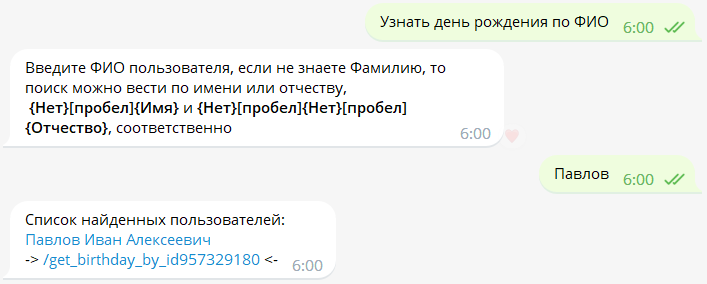


Рисунок – Дни рождения на ближайшие 14 дней



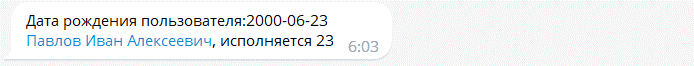


Рисунок – Поиск дня рождения по ФИО

Остальная функциональность была проверена и полностью соответствовала той, что была продемонстрирована у обычного пользователя.

7.1.4 Администратор

Администратором можно стать только после назначения таковым владельцем бота в специальной группе, в теме «Главная», после назначение в личный чат с пользователем бот присылает уведомление о том, что он теперь администратор и меняет главное меню на соответствующее, содержащее кнопки: «Мероприятия клуба», «Дни рождения». «Рассылка», «Инструкции», «Информация о пользователях», «Редактировать пользователя» (Рисунок 44).

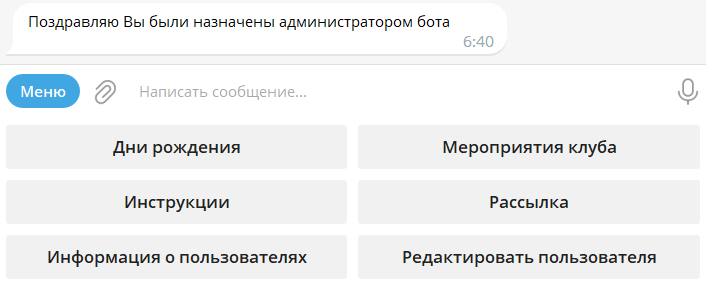


Рисунок – Получение прав администратора

Протестируем работу каждого пункта и подпункта меню.

Пункт «Дни рождения» абсолютно идентичен приведенному выше в роли пользователя+ и протестировав работает точно также.

При выборе пункта «Рассылка», бот спрашивает какое сообщение администратор планирует разослать всем пользователям (Рисунок 45), затем получив его, рассылает всем и закрепляет в личном чате (Рисунок 38).

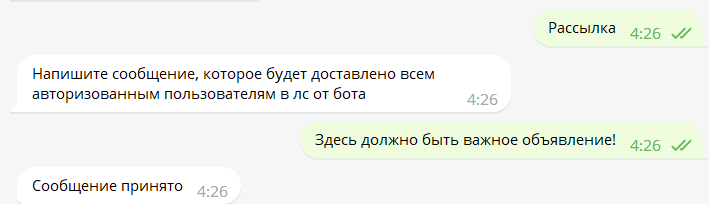


Рисунок – Формирование рассылки

При выборе пункта «Инструкции», бот присылает администратору инструкции работы с ботом в виде txt файла (Рисунок 46).

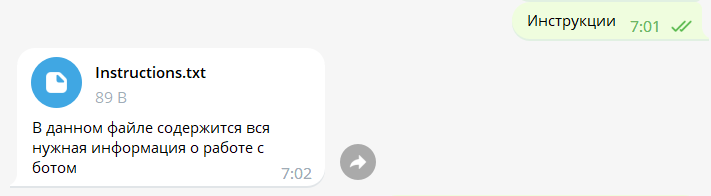


Рисунок – Получение инструкции работы с ботом

При выборе пункта «Мероприятия клуба», администратору открывается меню работы с мероприятиями клуба, состоящим из 3х пунктов: «Ближайшие мероприятия», «Отключить уведомления», «Создать мероприятие» (Рисунок 47).

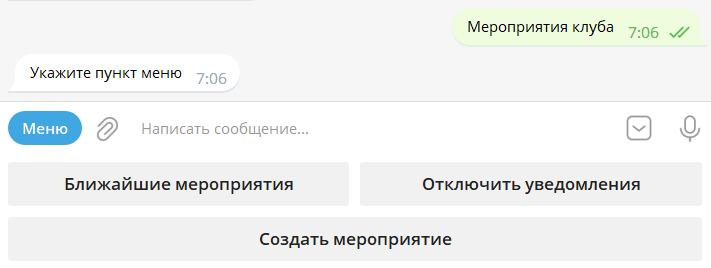


Рисунок – Меню системы мероприятий клуба

Пункт «Отключить уведомления» работает идентично тому, что есть у обычного пользователя (Рисунок 35).

Пункт «Ближайшие мероприятия» отличается от аналогичного пункта у обычного пользователя тем, что показывает все мероприятия и даже те, что уже прошли и возможностью удалить или по каждому атрибуту отредактировать каждое из мероприятий (Рисунок 48).

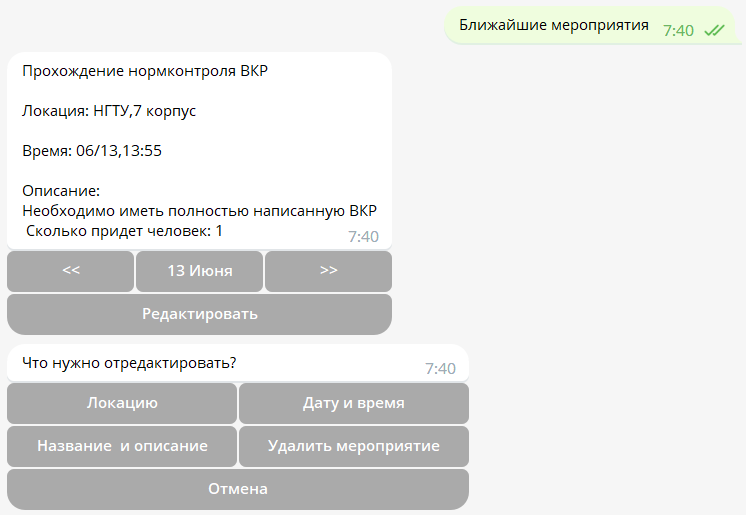
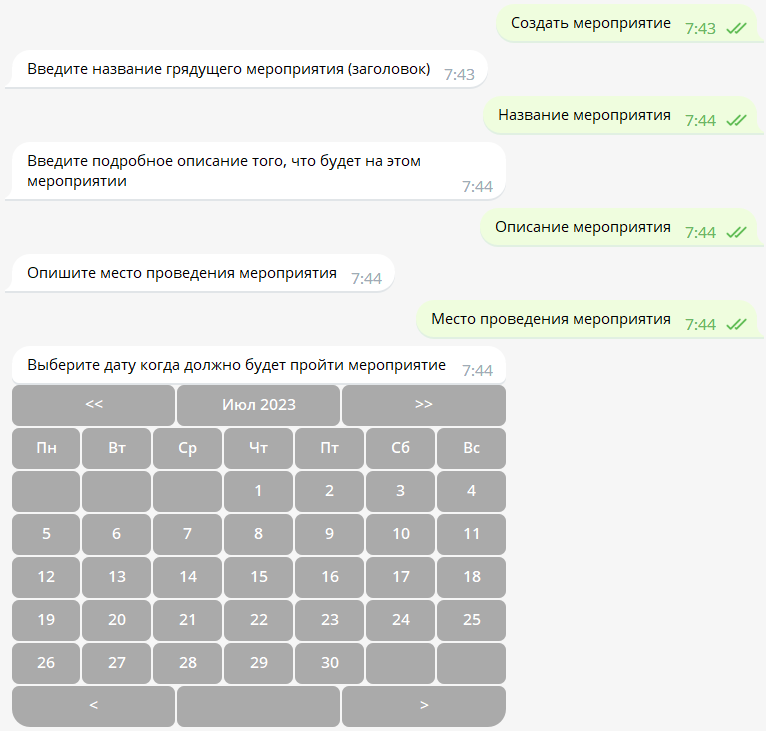
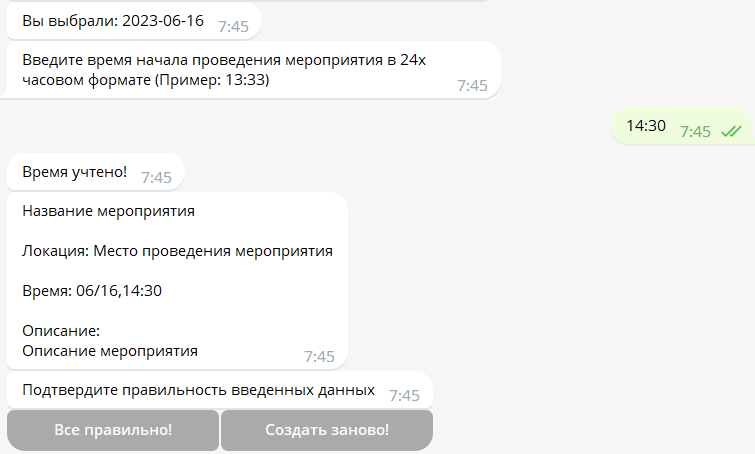


Рисунок – Просмотр и редактирование или удаление мероприятия

Пункт «Создать мероприятие» состоит из нескольких этапов (Рисунок 49):

* Ввод названия
* Описания
* Дата проведения, выбирается из предоставленного календаря
* Время начала, проверяется регулярным выражением на корректность
* Подтверждение правильности, если была допущена ошибка, то можно пересоздать

Рисунок – Создание мероприятия администратором



Пункт «Информация о пользователях» ведет к подменю с кнопками: «Найти пользователя по Гос. Номеру», «Найти пользователя по ФИО», «Получить данные о всех пользователях» (Рисунок 50). Протестируем каждый пункт подменю.

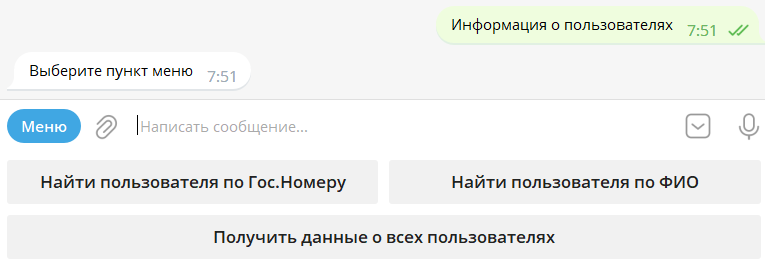


Рисунок – Меню выбора способа получение данных пользователей

При выборе подпункта «Найти пользователя по Гос. Номеру» у администратора запрашивается ввод гос. номера в свободном формате т.к. могут быть нестандартные номера (Рисунок 51) и выдается результат поиска (Рисунок 53).

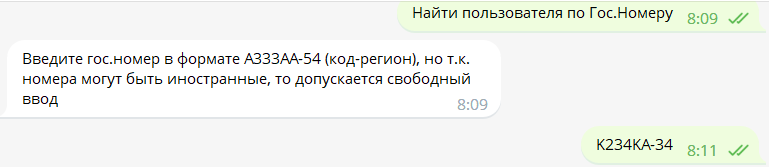


Рисунок – Ввод гос. номера искомого пользователя

При выборе подпункта «Найти пользователя по ФИО» у администратора запрашивается ввод ФИО с приложенной инструкцией по формату (Рисунок 51), после выдается список найденных по ФИО людей и команда под каждым запускает выдачу информации об искомом пользователе (Рисунок 53).

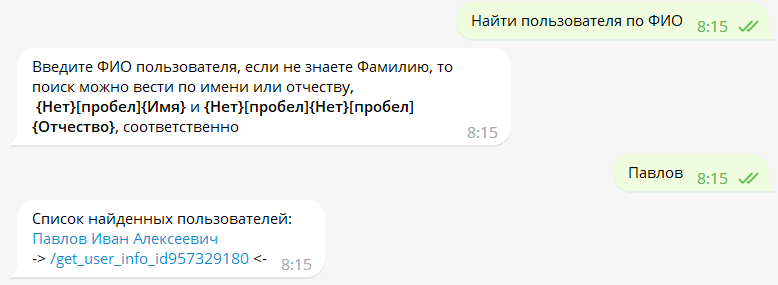


Рисунок – Получение информации о пользователе через ФИО

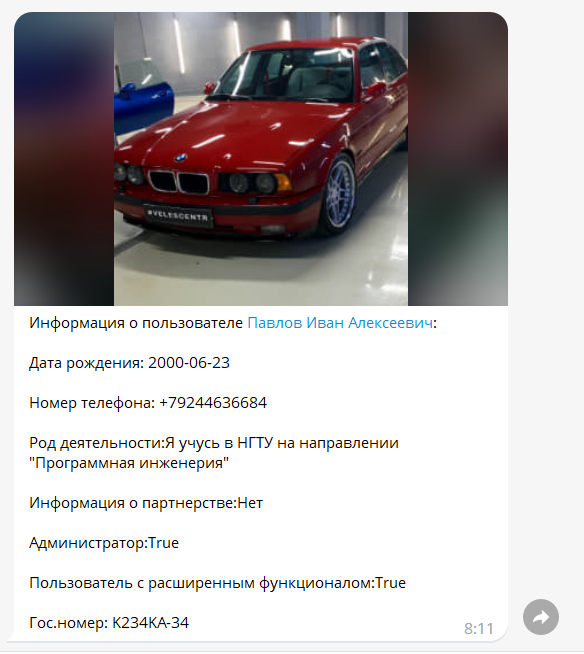


Рисунок – Полученная информация в результате поиска

При выборе подпункта «Получить данные о всех пользователях», бот высылает администратору excel файл со структурированной информацией на каждого пользователя (Рисунок 54).

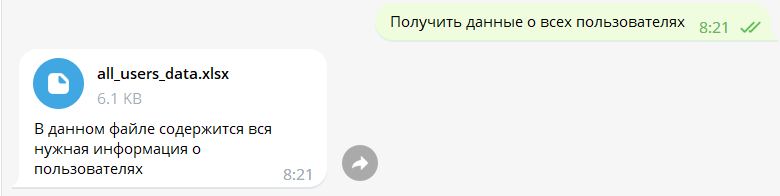


Рисунок – Получение всей информации о пользователях

При выборе пункта в главном меню «Редактировать пользователя», бот сразу запрашивает у администратора ФИО искомого пользователя (Рисунок 55).

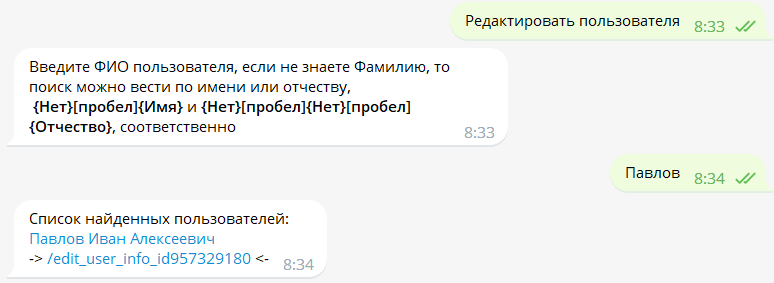


Рисунок – Редактировать пользователя по поиску через ФИО

После выбора команды под именем нужного человека, аналогично пункту выше, бот высылает информацию о пользователе (Рисунок 53) и сообщение о том, как и какие поля можно отредактировать (Рисунок 56).

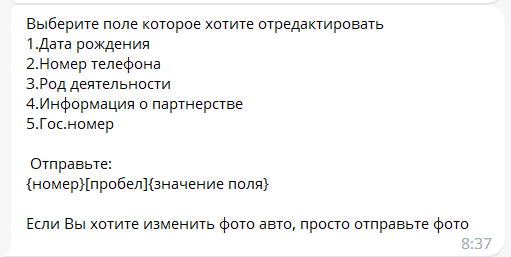


Рисунок – Пояснение какие поля и как можно редактировать у пользователя

Теперь проверим как работает назначение и удаление пользователей+, назначение пользователей+ происходит в специальной группе, в теме «Главная», командой /set\_new\_plus\_user, после запрашивается ФИО и выводится список найденных пользователей, после выбора определенного пользователя он становится пользователем+

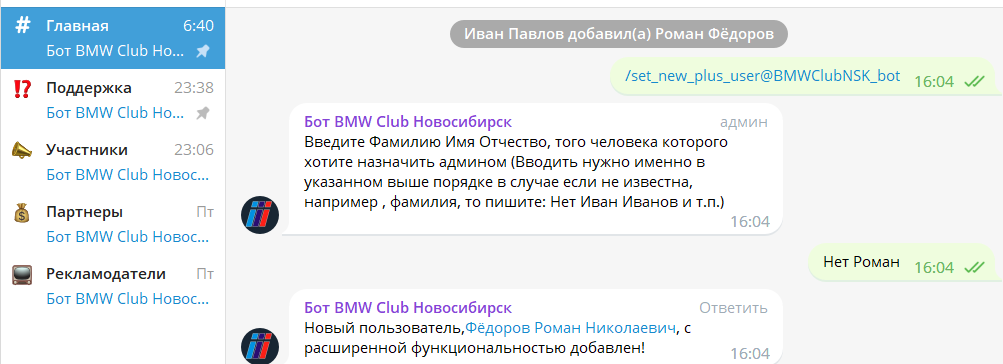
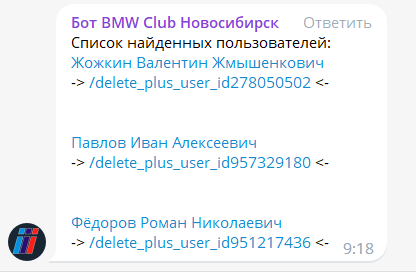


Рисунок -Пример применения команды добавления пользователя+

Удаление пользователя+ происходит после ввода команды /delete\_plus\_user, после чего присылается список людей являющимися пользователями+ с командами под каждым и после нажатия на команду нужный человек перестает быть пользователем+ (Рисунок 58)



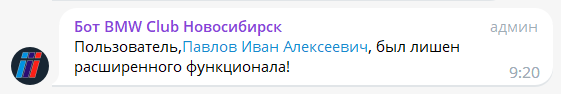
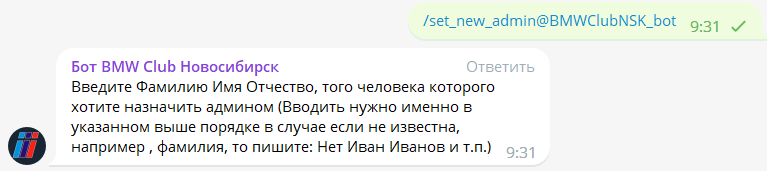
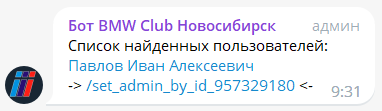


Рисунок – Список пользователей+ и удаление конкретного пользователя+

7.1.5 Владелец бота

Единственное различие между администратором и владельцем бота в том, что владелец имеет доступ к командам назначения и снятия администратора с должности в специальной группе, в теме «Главная», а именно: /set\_new\_admin (Рисунок 59) и /delete\_admin (Рисунок 60), протестируем их работу





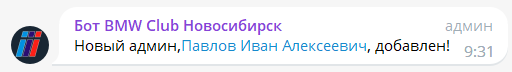


Рисунок – Добавление админа в специальной группе

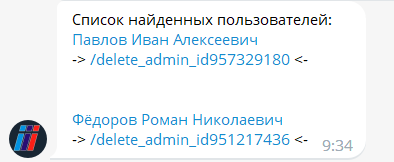


Рисунок – Снятие администратора с должности

7.2 Апробация

Работоспособность бота была проведена на нескольких ПК, но т.к. Telegram-бот является веб-приложением, то аппаратные характеристики для взаимодействия с Telegram-ботом для клиента ограничиваются требованиями desktop-приложения Telegram, а именно: [7]

* Процессор с тактовой частотой 800 MHz или более мощный
* Оперативная память 128 Мб или больше
* Свободное место на жёстком диске от 49 Мб
* Архитектура с разрядностью 32 бит или 64 бит (x86 или x64)
* Операционная система Windows XP, Windows Vista, Windows 7, Windows 8, Windows 10

А для сервера аппаратные характеристики ПК, с которого был запущен бот, будут ориентировочными, а именно:

* Процессор – Intel(R) Core (TM) i5-7300HQ CPU @ 2.50GHz х 4 ядра
* Оперативная память – 2x8 GB DDR4 4.066 GHz
* Операционная система – Windows 10 x64

В ходе апробации пользователями Telegram-бота технических недоработок выявлено не было.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы бакалавра был разработан Telegram-бот BMW Club Новосибирск, предназначенный для автоматизации процесса администрирования, информирования и обеспечения обратной связи с членами автоклуба. Поставленные задачи были выполнены в полном объёме.

* Произведен анализ аналогичных программных продуктов с целью выявления, достоинств и недостатков и востребованности той или иной разработки, которой посвящена работа. Так как полноценной альтернативы Telegram-боту, выполняющемуся на заказ не было выявлено, то анализ был проведен тех продуктов, что обладали частью той функциональности, что должен предоставлять бот. Итоги анализа были применены при составлении технического задания;
* Было проведено обоснование выбора средств разработки. В результате были выбраны язык Python, фреймворк AIOgram и СУБД PostgreSQL как наиболее соответствующие задачам инструменты;
* Требования к боту были определены в результате составления технического задания на основе данных, полученных при обсуждении целей, желаний и требований заказчика и проведение анализа сегмента аналогичных продуктов;
* Архитектура приложения была спроектирована на основе шаблона Model-View-Controller (MVC), что позволяет в будущем удобно масштабировать программный продукт и реализовывать новые функциональные возможности;
* На основе технического задания была сформирована диаграмма вариантов использования
* Функциональность была реализована с использованием AIOgram фреймворка и с применением асинхронного подхода;
* Этап контроля качества заключался в ручном тестировании;
* Успешное выполнение выпускной квалификационной работы бакалавра подтверждается Актом о внедрении результатов ВКРБ (приложение Б).

СПИСОК ЛИТЕРАТУРЫ

1. Most Loved, Dreaded, and Wanted Languages [Электронный ресурс] // StackOverflow. URL: <https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages> (дата обращения: 15.04.2023).

2. Why Python is Slow: Looking Under the Hood [Электронный ресурс] // Pythonic Perambulations. URL: <http://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/> (дата обращения: 19.04.2023).

3. PHP (Hypertext Preprocessor) [Электронный ресурс] // TechTarget. URL : <https://www.techtarget.com/whatis/definition/PHP-Hypertext-Preprocessor> (дата обращения: 19.04.2023).

4. Usage statistics of PHP for websites [Электронный ресурс] // W3Techs. URL: <https://w3techs.com/technologies/details/pl-php> (дата обращения: 20.04.2023).

5. aiogram documentation [Электронный ресурс] // aiogram. URL: <https://docs.aiogram.dev/en/dev-3.x/> (дата обращения: 25.04.2023).

6. About PostgreSQL [Электронный ресурс] // PostgreSQL URL: <https://www.postgresql.org/about/> (дата обращения: 13.05.2023).

7. Telegram [Электронный ресурс] // Systemreq URL: <https://systemreq.ru/telegram/> (дата обращения: 25.05.2023).

ПРИЛОЖЕНИЕ А. ЛИСТИНГ

**Run.py**

from bot import start\_bot  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 start\_bot()

**bot\main.py**

from aiogram import Bot  
from aiogram.contrib.fsm\_storage.memory import MemoryStorage  
from aiogram.types import ParseMode  
from aiogram.utils import executor  
  
from bot.commands import register\_all\_commands  
from bot.database import database as conn  
from bot.database.methods.database\_polling import start\_process\_db\_polling  
from bot.handlers.main import register\_all\_handlers  
from bot.misc.log\_middleware import \*  
  
  
async def \_\_on\_start\_up(dp: Dispatcher) -> None:  
 conn.start()  
 register\_all\_filters(dp)  
 await register\_all\_commands(dp)  
 register\_all\_handlers(dp)  
 await start\_process\_db\_polling(dp.bot)  
  
  
def start\_bot():  
 bot: Bot = Bot(token=Env.TOKEN, parse\_mode=ParseMode.HTML)  
 dp = Dispatcher(bot, storage=MemoryStorage())  
 dp.setup\_middleware(LogMiddleware())  
  
 executor.start\_polling(dp, skip\_updates=True, on\_startup=\_\_on\_start\_up)

**bot/env.py**

import os  
from abc import ABC  
from typing import Final  
  
  
class Env(ABC):  
 TOKEN: Final = os.environ.get('TOKEN', '6210427421:AAEpZGEeuxV18QOJO18c5rYp3pxzFXK0z0')  
 NOTIFICATION\_SUPER\_GROUP\_ID = os.environ.get('NOTIF\_GROUP\_ID', -1001813689601)  
 QUESTION\_THREAD\_ID = os.environ.get('QUESTION\_THREAD\_ID', 6)  
 NEW\_MEMBER\_THREAD\_ID = os.environ.get('NEW\_MEMBER\_THREAD\_ID', 8)  
 NEW\_PARTNER\_THREAD\_ID = os.environ.get('NEW\_PARTNER\_THREAD\_ID', 10)  
 NEW\_ADS\_THREAD\_ID = os.environ.get('NEW\_ADS\_THREAD\_ID', 438)  
 SECRET\_PASSWORD = os.environ.get('SECRET\_PASSWORD', '3487bmw\_club-passw037tyfggfdsz')  
 CONNECTION\_URL = os.environ.get('CONNECTION\_URL', 'postgresql://postgres:1824@localhost:5432/BMW\_Club\_NSK')

**bot/commands.py**

from aiogram import Dispatcher  
from aiogram.types import \*  
  
from bot.env import Env  
  
  
async def register\_all\_commands(dp: Dispatcher):  
 await dp.bot.set\_my\_commands(  
 [BotCommand('set\_new\_plus\_user', "Назначение пользователя с расширенным функционалом "),  
 BotCommand('set\_new\_admin', "Назначение пользователя админом "),  
 BotCommand('delete\_plus\_user', "Удаление пользователя с расширенным функционалом "),  
 BotCommand('delete\_admin', "Удаление админа ")  
 ],  
 BotCommandScopeChat(Env.NOTIFICATION\_SUPER\_GROUP\_ID))  
 await dp.bot.set\_my\_commands(  
 [BotCommand('main\_menu', "Возврат в главное меню \n (Только для зарегистрированных пользователей)")],  
 BotCommandScopeAllPrivateChats())

**bot/res/strings.py**

start\_response = """Вас приветствует бот BMW Club Новосибирск!  
  
BMWClubNovosibirsk - не просто группа или чат) мы общаемся, помогаем друг другу в самых разных ситуациях и сферах жизни, периодически встречаемся (кафе,кальянная,выезд на природу, аренда коттеджей и пр), занимаемся благотворительностью и т.д.  
  
Если Вы хотите стать частью нашего клуба, то Вам необходимо будет пройти регистрацию, где Вам необходимо будет указать следующую информацию:  
 1.Ваше Имя, Фамилию и Отчество (если есть)  
 2.Дата рождения  
 3.Профессия/род деятельности/ чем и   
 в чём сможете помочь советом или делом  
 4.Фото авто (Мы авто-клуб как никак 😄)  
 5.Гос.номер  
 6.Номер телефона  
 7.Страницы, сайты, номера тел и прочее,   
 если вы или ваши знакомые хотят стать   
 партнёрами клуба, дать скидку и тд.   
 в ЛЮБОЙ из сфер жизни.   
   
Подробности у https://t.me/Polina\_414.  
Данная информация будет СКРЫТА и доступна только ответственному кругу лиц и администратору клуба.  
  
Если вы хотите стать нашим партнером или рекламодателем, то нажмите на соответствующие кнопки ниже и мы с Вами свяжемся   
"""  
  
registr\_attention = "Пожалуйста, указывайте свои данные без ошибок, иначе процесс регистрации придется проходить заново"  
registr\_attention\_partner = "Вы хотите стать нашим партнером. После заполнения небольшой формы мы свяжемся с Вами," \  
 " чтобы обсудить детали партнерства."  
registr\_attention\_ads = "Вы хотите стать нашим рекламодателем. После заполнения небольшой формы мы свяжемся с Вами," \  
 " чтобы обсудить детали и условия размещения рекламы."  
input\_surname = "Укажите Вашу Фамилию"  
input\_name = "Укажите Ваше Имя"  
input\_patronymic = 'Укажите Ваше Отчество, если его нет, то напишите “Нет”'  
input\_birthday = 'Укажите дату Вашего рождения'  
input\_about = 'Расскажите о себе, чем занимаетесь/ род деятельности/ хобби или в чем разбираетесь и в случае чего к ' \  
 'Вам можно было обратиться за советом. Например, "Я работаю адвокатом, разбираюсь в административном праве"'  
input\_car\_photo = 'Загрузите фото вашего авто'  
input\_car\_number\_plate = '''Введите гос.номер вашего авто, в формате: С065МК-78  
Гос.номер-Код региона '''  
err\_input\_number\_plate = '''Вы написали номер с ошибкой!  
Введите номер заново'''  
input\_non\_rus\_car\_number\_plate = 'Введите гос.номер Вашего авто'  
input\_phone\_number = 'Отправьте Ваш номер телефона, нажав кнопку ниже'  
partner\_choice = '''Хотите ли Вы стать партнером нашего клуба?   
  
Страницы, сайты, номера тел и прочее, если Вы или ваши знакомые хотят стать партнёрами клуба, дать скидку и тд. в ЛЮБОЙ из сфер жизни.   
Например, если Вы владелец автосервиса, то можете стать нашим партнером получив в замен крупный и постоянный поток клиентов и т.п.   
Подробности у https://t.me/Polina\_414.  
'''  
about\_partner\_info = '''Расскажите о своем деле, том чем мы и Ваш бизнес могли бы сотрудничать, например скидки, услуги или какой либо ещё вид помощи связанный с вашим бизнесом или организацией.   
  
Позже с Вами свяжутся администраторы, чтобы более подробно обсудить детали сотрудничества.'''  
about\_ads\_info = '''Расскажите о своей организации и о том какую бы рекламу хотели бы у нас разместить.   
  
Позже с Вами свяжутся администраторы, чтобы более подробно обсудить детали сотрудничества.'''  
business\_info\_answer = '''Ожидайте скоро наши администраторы свяжутся с вами  
Если хотите пройти регистрацию заново нажмите /start'''  
end\_registration = 'Если Вы уверены в правильности введённых данных нажмите кнопку “Завершить регистрацию”'  
finish\_registration = 'Регистрация была успешно завершена!'  
  
registration\_help = 'Расскажите о том, с чем у Вас возникают трудности.Опишите свою проблему или вопрос и скоро с Вами свяжутся администраторы.'  
registration\_help\_answer = '''Ваш вопрос принят.  
Ждите скоро наши администраторы свяжутся с вами  
Для того чтобы пройти регистрацию заново нажмите /start'''

**bot/misc/states.py**

*# тут содержатся состояния для машины состояний*from typing import Final  
  
from aiogram.dispatcher.filters.state import StatesGroup, State  
  
  
class RegisterUser(StatesGroup):  
 INSERT\_SURNAME: Final = State() *# Состояние при котором принимается Фамилия* INSERT\_NAME: Final = State() *# Состояние при котором принимается Имя* INSERT\_PATRONYMIC: Final = State() *# Состояние при котором принимается Отчество* INSERT\_ABOUT: Final = State() *# Состояние при котором принимается о том чем занимается пользователь* INSERT\_CAR\_PHOTO: Final = State() *# Состояние при котором принимается Фото Авто* INSERT\_NUMBER\_PLATE: Final = State() *# Состояние при котором принимается Гос. Номер Авто* INSERT\_NON\_RUS\_PLATE: Final = State() *# Состояние при котором принимается иностранный Гос. Номер Авто* INSERT\_PARTNER\_BUSINESS: Final = State() *# Состояние при котором принимается Информация о партнерстве* INSERT\_ABOUT\_BUSINESS: Final = State()  
 NEUTRAL: Final = State()  
 HELP\_QUESTION: Final = State()  
  
  
class CreateEvent(StatesGroup):  
 INSERT\_TITLE: Final = State() *# Состояние при котором принимается заголовок события* INSERT\_DESCRIPTION: Final = State() *# Состояние при котором принимается описание* INSERT\_LOCATION: Final = State() *# Состояние при котором принимается место проведения* INSERT\_DATA: Final = State() *# Состояние при котором принимается дата проведения* INSERT\_TIME: Final = State() *# Состояние при котором принимается время проведения* FINISH\_CREATE: Final = State() *# Состояние при котором должно завершаться создание мероприятия*class EditEvent(StatesGroup):  
 INSERT\_TITLE\_DESCRIPTION: Final = State() *# Состояние при котором принимается описание* INSERT\_LOCATION: Final = State() *# Состояние при котором принимается место проведения* INSERT\_DATA\_TIME: Final = State() *# Состояние при котором принимается время проведения*class UpdatePermissions(StatesGroup):  
 INSERT\_ADMIN\_FIO: Final = State() *# Состояние при котором принимается ФИО админа* CHOICE\_ADMIN: Final = State() *# Состояние при котором выбираться из найденных* INSERT\_USER\_PLUS\_FIO: Final = State() *# Состояние при котором принимается ФИО пользователя* CHOICE\_USER\_PLUS: Final = State()  
  
  
class AdminStates(StatesGroup):  
 INSERT\_BROADCAST: Final = State() *# Ввод рассылочного сообщения* INSERT\_USER\_FIO: Final = State() *# Ввод ФИО искомого пользователя* INSERT\_USER\_PLATE: Final = State()  
 CHOICE\_USER\_INFO: Final = State()  
 EDIT\_USER: Final = State()  
 CHOICE\_USER\_EDIT: Final = State()  
 EDIT\_DATA\_USER: Final = State()  
 INSERT\_USER\_FIO\_BIRTHDAY: Final = State() *# Ввод ФИО искомого пользователя* GET\_BIRTHDAY\_FIO: Final = State()

**bot/misc/log\_middleware.py**

from typing import Callable, Dict, Any, Awaitable  
  
from aiogram.dispatcher import FSMContext  
from aiogram.dispatcher.handler import current\_handler, CancelHandler  
from aiogram.dispatcher.middlewares import BaseMiddleware  
from aiogram.types import Message, Update  
from bot.filters.main import \*  
from bot.handlers.user import main as regular\_user  
from bot.handlers.user import plus\_user  
from bot.handlers.admin import main as adm  
from bot.handlers.registration import main as registr  
  
  
class LogMiddleware(BaseMiddleware):  
 def \_\_init\_\_(self) -> None:  
 self.counter = 0  
 super(LogMiddleware, self).\_\_init\_\_()  
  
 async def on\_process\_message(  
 self,  
 message: Message,  
 data: Dict[str, Any]  
 ) -> Any:  
 self.counter += 1  
 state = data.get("state")  
 handler = current\_handler.get()  
 if message.text == '/main\_menu':  
 if await IsAdminOrOwner().check(message):  
 await adm.get\_menu(message=message, state=state)  
 if await IsRegularUserOnly().check(message):  
 await regular\_user.get\_menu(message=message, state=state)  
 if await IsPlusUserOnly().check(message):  
 await plus\_user.get\_menu(message=message, state=state)  
 if await IsNotRegistered().check(message):  
 await registr.get\_menu(message, state)  
 raise CancelHandler()  
  
 print(handler.\_\_name\_\_)  
 *# return await handler(event, data)*

**bot/misc/formatting.py**

import datetime  
  
  
async def format\_birthday(birthdays, day) -> str:  
 text: str = 'В эту дату\, день рождения у:'  
 if len(birthdays[day][0]) != 0:  
 for birthday in birthdays[day][0]: *# [номер дня][0 - пользователи , 1 - дата (месяц и день)]* text += '\n[{} {} {}](tg://user?id={})\, исполняется {} '.format(birthday[1],  
 birthday[2],  
 birthday[3],  
 birthday[0],  
 birthday[4])  
 else:  
 text = 'В этот день ни у кого дня рождения нет\!'  
 return text  
  
  
async def format\_birthday\_by\_fio(birthday) -> str:  
 text: str = 'Дата рождения пользователя:{}'.format(birthday[0][5].strftime('%Y\-%m\-%d'))  
 if len(birthday) != 0:  
 text += '\n[{} {} {}](tg://user?id={})\, исполняется {} '.format(birthday[0][1],  
 birthday[0][2],  
 birthday[0][3],  
 birthday[0][0],  
 birthday[0][4])  
 else:  
 text = 'В этот день ни у кого дня рождения нет\!'  
 return text  
  
  
async def replace\_markdown\_marks(string: str):  
 strings\_replace = ['\_', '\*', '[', ']', '(', ')', '~', '`', '>',  
 '#', '+', '-', '=', '|', '{', '}', '.', '!']  
 for symbol in strings\_replace: *# Экранируем спец символы для ТГ-разметки сообщений* string = string.replace(symbol, '\\' + symbol)  
 return string  
  
  
async def format\_founded\_users(users: list, command: str):  
 text: str = 'Список найденных пользователей:'  
 users\_fio\_id: dict = {}  
 if len(users) == 0:  
 text += '\nПуст'  
 for user in users:  
 user\_link = '[{} {} {}](tg://user?id={})'.format(user[1],  
 user[2],  
 user[3],  
 user[0])  
 text += '\n' + user\_link + ' \n\-\> {}{} \<\-\n\n'.format(command, user[0])  
 users\_fio\_id[str(user[0])] = user\_link  
 return text, users\_fio\_id  
  
  
async def format\_event(title: str, date: datetime.datetime, time: datetime.time | str, location: str,  
 description: str) -> str:  
 text: str = title + '\n'  
 text += '\nЛокация: ' + location  
 text += '\n\nВремя: ' + date.strftime('%m/%d')  
 if type(time) == datetime.time:  
 text += ',' + time.strftime('%H:%M')  
 else:  
 text += ',' + time  
 text += '\n\nОписание:\n' + description  
 return text  
  
  
async def format\_event\_extended(title: str, date: datetime.datetime, time: datetime.time | str, location: str,  
 description: str, amount\_users) -> str:  
 text: str = await format\_event(title, date, time, location, description)  
 text += '\n Сколько придет человек: ' + str(amount\_users)  
 return text  
  
  
async def user\_info(user):  
 birthday: datetime.date = user[4]  
  
 text: str = 'Информация о пользователе <a href="tg://user?id={}">{} {} {}</a>:\n' \  
 "\nДата рождения: {}\n" \  
 "\nНомер телефона: {}\n" \  
 "\nРод деятельности:{}\n" \  
 "\nИнформация о партнерстве:{}\n" \  
 "\nАдминистратор:{}\n" \  
 "\nПользователь с расширенным функционалом:{}\n" \  
 "\nГос.номер: {}\n" \  
 "".format(user[0], user[1], user[2], user[3], birthday.strftime('%Y-%m-%d'),  
 user[5], user[6], user[7], user[8], user[9], user[10])  
 return text, user[11] *# Описание пользователя и фото авто*

**bot/misc/anti\_swearing.py**

*# Символы, что должны экранироваться*strings\_replace = ['\_', '\*', '[', ']', '(', ')', '~', '`', '>',  
 '#', '+', '-', '=', '|', '{', '}', '.', '!']  
  
mat = [  
*#элементы нецензурной лексики* ]  
  
marks = '''!()-[]{};?@#$%:'"\,./^&amp;\*\_'''  
  
  
async def censoring\_text(string: str):  
 string = string.lower()  
 string = " " + string + " "  
 for x in string:  
 if x in marks:  
 string = string.replace(x, " " + x)  
 for symbol in strings\_replace: *# Экранируем спец символы для ТГ-разметки сообщений* string = string.replace(symbol, '\\' + symbol)  
 for bad\_word in mat: *# Скрываем плохие слова в сообщении* bad\_word = " " + bad\_word + " "  
 string = string.replace(bad\_word, "||" + bad\_word + "||")  
 return string  
  
  
async def is\_swearing(string: str) -> bool:  
 string = string.lower()  
 test = " " + string + " "  
 for x in string:  
 if x in marks:  
 test = test.replace(x, " " + x)  
 for bad\_word in mat: *# Скрываем плохие слова в сообщении* bad\_word = " " + bad\_word + " "  
 *# bad\_word = bad\_word.translate({ord(' '): None})* if bad\_word in test:  
 return True  
 return False

**bot/keyboards/simple\_calendar.py**

import calendar  
from datetime import datetime, timedelta  
  
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton  
from aiogram.utils.callback\_data import CallbackData  
from aiogram.types import CallbackQuery  
  
*# setting callback\_data prefix and parts*simple\_calendar\_callback = CallbackData('simple\_calendar', 'act', 'year', 'month', 'day')  
  
  
class SimpleCalendar:  
  
 async def start\_calendar(self, year: int = datetime.now().year,  
 month: int = datetime.now().month) -> InlineKeyboardMarkup:  
 inline\_kb = InlineKeyboardMarkup(row\_width=7)  
 ignore\_callback = simple\_calendar\_callback.new("IGNORE", year, month, 0) *# for buttons with no answer  
 # First row - Month and Year* month\_name = ["Янв", "Фев", "Мар", "Апр", "Май", "Июн", "Июл", "Авг", "Сен", "Окт", "Ноя", "Дек"]  
  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 "<<",  
 callback\_data=simple\_calendar\_callback.new("PREV-YEAR", year, month, 1)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 f'{month\_name[month]} {str(year)}',  
 callback\_data=ignore\_callback  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 ">>",  
 callback\_data=simple\_calendar\_callback.new("NEXT-YEAR", year, month, 1)  
 ))  
 *# Second row - Week Days* inline\_kb.row()  
 for day in ["Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Вс"]:  
 inline\_kb.insert(InlineKeyboardButton(day, callback\_data=ignore\_callback))  
  
 *# Calendar rows - Days of month* month\_calendar = calendar.monthcalendar(year, month)  
 for week in month\_calendar:  
 inline\_kb.row()  
 for day in week:  
 if day == 0:  
 inline\_kb.insert(InlineKeyboardButton(" ", callback\_data=ignore\_callback))  
 continue  
 inline\_kb.insert(InlineKeyboardButton(  
 str(day), callback\_data=simple\_calendar\_callback.new("DAY", year, month, day)  
 ))  
  
 *# Last row - Buttons* inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 "<", callback\_data=simple\_calendar\_callback.new("PREV-MONTH", year, month, day)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(" ", callback\_data=ignore\_callback))  
 inline\_kb.insert(InlineKeyboardButton(  
 ">", callback\_data=simple\_calendar\_callback.new("NEXT-MONTH", year, month, day)  
 ))  
  
 return inline\_kb  
  
 async def process\_selection(self, query: CallbackQuery, data: CallbackData) -> tuple:  
 *"""  
 Process the callback\_query. This method generates a new calendar if forward or  
 backward is pressed. This method should be called inside a CallbackQueryHandler.  
 :param query: callback\_query, as provided by the CallbackQueryHandler  
 :param data: callback\_data, dictionary, set by calendar\_callback  
 :return: Returns a tuple (Boolean,datetime), indicating if a date is selected  
 and returning the date if so.  
 """* return\_data = (False, None)  
 temp\_date = datetime(int(data['year']), int(data['month']), 1)  
 *# processing empty buttons, answering with no action* if data['act'] == "IGNORE":  
 await query.answer(cache\_time=60)  
 *# user picked a day button, return date* if data['act'] == "DAY":  
 await query.message.delete\_reply\_markup() *# removing inline keyboard* return\_data = True, datetime(int(data['year']), int(data['month']), int(data['day']))  
 *# user navigates to previous year, editing message with new calendar* if data['act'] == "PREV-YEAR":  
 prev\_date = datetime(int(data['year']) - 1, int(data['month']), 1)  
 await query.message.edit\_reply\_markup(await self.start\_calendar(int(prev\_date.year), int(prev\_date.month)))  
 *# user navigates to next year, editing message with new calendar* if data['act'] == "NEXT-YEAR":  
 next\_date = datetime(int(data['year']) + 1, int(data['month']), 1)  
 await query.message.edit\_reply\_markup(await self.start\_calendar(int(next\_date.year), int(next\_date.month)))  
 *# user navigates to previous month, editing message with new calendar* if data['act'] == "PREV-MONTH":  
 prev\_date = temp\_date - timedelta(days=1)  
 await query.message.edit\_reply\_markup(await self.start\_calendar(int(prev\_date.year), int(prev\_date.month)))  
 *# user navigates to next month, editing message with new calendar* if data['act'] == "NEXT-MONTH":  
 next\_date = temp\_date + timedelta(days=31)  
 await query.message.edit\_reply\_markup(await self.start\_calendar(int(next\_date.year), int(next\_date.month)))  
 *# at some point user clicks DAY button, returning date* return return\_data

**bot/keyboards/reply.py**

from aiogram.types import ReplyKeyboardMarkup, KeyboardButton  
  
  
async def get\_phone\_number() -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True, one\_time\_keyboard=True)  
 kb.row()  
 kb.add(KeyboardButton('Отправить номер телефона', request\_contact=True))  
 return kb  
  
  
async def registration\_menu() -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True, is\_persistent=True)  
 kb.row()  
 kb.add(KeyboardButton('Получить помощь по регистрации'))  
 kb.add(KeyboardButton('Допустил ошибку/Начать заново'))  
 return kb  
  
  
async def regular\_user\_start\_menu() -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True, is\_persistent=True)  
 kb.row(KeyboardButton('Получить помощь/Задать вопрос'), KeyboardButton('Мероприятия клуба'))  
 return kb  
  
  
async def plus\_user\_start\_menu() -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True, is\_persistent=True)  
 kb.row(KeyboardButton('Дни рождения'), KeyboardButton('Мероприятия клуба'))  
 kb.row(KeyboardButton('Получить помощь/Задать вопрос'))  
 return kb  
  
  
async def admin\_menu() -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True, is\_persistent=True)  
 kb.row(KeyboardButton('Дни рождения'), KeyboardButton('Мероприятия клуба'))  
 kb.row(KeyboardButton('Инструкции'), KeyboardButton('Рассылка'))  
 kb.row(KeyboardButton('Информация о пользователях'), KeyboardButton('Редактировать пользователя'))  
 return kb  
  
  
async def birthday\_menu(notif\_on) -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True)  
 kb.row(KeyboardButton('Показать ДР на ближайшие 14 дней'),  
 KeyboardButton('Отключить уведомления о ДР' if notif\_on[0] else 'Включить уведомления о ДР')  
 )  
 kb.row(KeyboardButton('Узнать день рождения по ФИО'))  
 return kb  
  
  
async def event\_admin\_menu(notif\_on) -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True)  
 kb.row(KeyboardButton('Ближайшие мероприятия'),  
 KeyboardButton('Отключить уведомления' if notif\_on[0] else 'Включить уведомления'))  
 kb.row(KeyboardButton('Создать мероприятие'))  
 return kb  
  
  
async def event\_menu(notif\_on) -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True)  
 kb.row(KeyboardButton('Ближайшие мероприятия'),  
 KeyboardButton('Отключить уведомления' if notif\_on[0] else 'Включить уведомления'))  
 return kb  
  
  
async def users\_info\_menu() -> ReplyKeyboardMarkup:  
 kb = ReplyKeyboardMarkup(resize\_keyboard=True)  
 kb.row(  
 KeyboardButton('Найти пользователя по Гос.Номеру'),  
 KeyboardButton('Найти пользователя по ФИО')  
 )  
 kb.row(KeyboardButton('Получить данные о всех пользователях'))  
 return kb

**bot/keyboards/inline.py**

from aiogram.types import \*  
from aiogram.utils.callback\_data import CallbackData  
  
  
async def start\_reg\_message() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=2)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Я бизнес-партнер", callback\_data='register\_partner'))  
 kb.insert(InlineKeyboardButton("Я рекламодатель", callback\_data='register\_ads'))  
 kb.row()  
 kb.insert(InlineKeyboardButton("Я участник клуба", callback\_data='register'))  
 return kb  
  
  
async def car\_number\_plate\_non\_rus() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=1)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Мой гос.номер другого формата", callback\_data='non\_rus\_plate'))  
 return kb  
  
  
async def partner\_choice() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=2)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Я хочу стать партнёром", callback\_data='wanna\_be\_partner'))  
 kb.insert(InlineKeyboardButton("Нет", callback\_data='no\_partner'))  
 return kb  
  
  
async def end\_registration() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=1)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Завершить регистрацию", callback\_data='end\_registration'))  
 return kb  
  
  
question\_answer\_callbackdata = CallbackData('answer', 'user\_id')  
  
  
async def question\_answer(user\_id) -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=2)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Ответить на вопрос", url="tg://user?id={}".format(user\_id)))  
 kb.insert(InlineKeyboardButton("Отвечен!", callback\_data=question\_answer\_callbackdata.new(user\_id)))  
 return kb  
  
  
async def question\_delete() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=1)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Удалить вопрос",  
 callback\_data='delete\_question'))  
 return kb  
  
  
async def end\_create\_event() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=2)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Все правильно!", callback\_data="correct\_event"))  
 kb.insert(InlineKeyboardButton("Создать заново!", callback\_data='recreate\_event'))  
 return kb  
  
  
async def edit\_event() -> InlineKeyboardMarkup:  
 kb = InlineKeyboardMarkup(row\_width=2)  
 kb.row()  
 kb.insert(InlineKeyboardButton("Локацию", callback\_data="location\_edit"))  
 kb.insert(InlineKeyboardButton("Дату и время", callback\_data='data/time\_edit'))  
 kb.row()  
 kb.insert(InlineKeyboardButton("Название и описание", callback\_data="title\_and\_desc\_edit"))  
 kb.insert(InlineKeyboardButton("Удалить мероприятие", callback\_data='delete\_event'))  
 kb.row()  
 kb.insert(InlineKeyboardButton("Отмена", callback\_data='cancel\_event'))  
 return kb

**bot/keyboards/events\_slider\_admin.py**

import datetime  
  
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton, CallbackQuery  
from aiogram.utils.callback\_data import CallbackData  
  
from bot.misc.formatting import \*  
  
event\_slider\_admin\_callback = CallbackData('slider\_event\_admin', 'action\_event', 'index')  
  
  
class EventsSliderAdmin:  
 def \_\_init\_\_(self, events):  
 self.events = events  
  
 async def get\_slider\_markup(self, index: int = 0) -> InlineKeyboardMarkup:  
 months = ["Января", "Февраля", "Марта", "Апреля", "Мая", "Июня",  
 "Июля", "Августа", "Сентября", "Октября", "Ноября", "Декабря"]  
 date: datetime.datetime = self.events[index][2]  
 inline\_kb = InlineKeyboardMarkup(row\_width=3)  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 '<<',  
 callback\_data=event\_slider\_admin\_callback.new("PREV", (index - 1))  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '{} {}'.format(date.day, months[int(date.month) - 1]),  
 callback\_data=event\_slider\_admin\_callback.new("IGNORE", index)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '>>',  
 callback\_data=event\_slider\_admin\_callback.new("NEXT", (index + 1))  
 ))  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 'Редактировать',  
 callback\_data=event\_slider\_admin\_callback.new("EDIT", index)  
 ))  
 return inline\_kb  
  
 async def new\_text(self, index: int) -> str:  
 return await format\_event\_extended(self.events[index][1], self.events[index][2], self.events[index][3],  
 self.events[index][4], self.events[index][5], str(self.events[index][6]))  
  
 async def selection(self, query: CallbackQuery, callback\_data: dict) -> tuple:  
 return\_data: tuple = (False, None)  
 if callback\_data['action\_event'] == "IGNORE":  
 await query.answer(cache\_time=60)  
 if callback\_data['action\_event'] == "PREV":  
 if int(callback\_data['index']) >= 0:  
 await query.message.edit\_text(await self.new\_text(int(callback\_data['index'])), 'HTML',  
 reply\_markup=await self.get\_slider\_markup(int(callback\_data['index'])))  
 if callback\_data['action\_event'] == "NEXT":  
 if int(callback\_data['index']) <= len(self.events) - 1:  
 await query.message.edit\_text(await self.new\_text(int(callback\_data['index'])), 'HTML',  
 reply\_markup=await self.get\_slider\_markup(int(callback\_data['index'])))  
  
 *# В случае если админ хочет отредактировать событие то функция возвратит id события* if callback\_data['action\_event'] == "EDIT":  
 return\_data = True, self.events[int(callback\_data['index'])][0]  
 return return\_data

**bot/keyboards/events\_slider.py**

import datetime  
  
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton, CallbackQuery  
from aiogram.utils.callback\_data import CallbackData  
  
from bot.misc.formatting import \*  
  
event\_slider\_admin\_callback = CallbackData('slider\_event\_admin', 'action\_event', 'index')  
  
  
class EventsSliderAdmin:  
 def \_\_init\_\_(self, events):  
 self.events = events  
  
 async def get\_slider\_markup(self, index: int = 0) -> InlineKeyboardMarkup:  
 months = ["Января", "Февраля", "Марта", "Апреля", "Мая", "Июня",  
 "Июля", "Августа", "Сентября", "Октября", "Ноября", "Декабря"]  
 date: datetime.datetime = self.events[index][2]  
 inline\_kb = InlineKeyboardMarkup(row\_width=3)  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 '<<',  
 callback\_data=event\_slider\_admin\_callback.new("PREV", (index - 1))  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '{} {}'.format(date.day, months[int(date.month) - 1]),  
 callback\_data=event\_slider\_admin\_callback.new("IGNORE", index)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '>>',  
 callback\_data=event\_slider\_admin\_callback.new("NEXT", (index + 1))  
 ))  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 'Редактировать',  
 callback\_data=event\_slider\_admin\_callback.new("EDIT", index)  
 ))  
 return inline\_kb  
  
 async def new\_text(self, index: int) -> str:  
 return await format\_event\_extended(self.events[index][1], self.events[index][2], self.events[index][3],  
 self.events[index][4], self.events[index][5], str(self.events[index][6]))  
  
 async def selection(self, query: CallbackQuery, callback\_data: dict) -> tuple:  
 return\_data: tuple = (False, None)  
 if callback\_data['action\_event'] == "IGNORE":  
 await query.answer(cache\_time=60)  
 if callback\_data['action\_event'] == "PREV":  
 if int(callback\_data['index']) >= 0:  
 await query.message.edit\_text(await self.new\_text(int(callback\_data['index'])), 'HTML',  
 reply\_markup=await self.get\_slider\_markup(int(callback\_data['index'])))  
 if callback\_data['action\_event'] == "NEXT":  
 if int(callback\_data['index']) <= len(self.events) - 1:  
 await query.message.edit\_text(await self.new\_text(int(callback\_data['index'])), 'HTML',  
 reply\_markup=await self.get\_slider\_markup(int(callback\_data['index'])))  
  
 *# В случае если админ хочет отредактировать событие то функция возвратит id события* if callback\_data['action\_event'] == "EDIT":  
 return\_data = True, self.events[int(callback\_data['index'])][0]  
 return return\_data

**bot/keyboards/date\_picker.py**

import calendar  
from datetime import datetime  
  
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton  
from aiogram.utils.callback\_data import CallbackData  
from aiogram.types import CallbackQuery  
  
date\_picker\_callback = CallbackData('date\_picker', 'act', 'year', 'month', 'day')  
ignore\_callback = date\_picker\_callback.new("IGNORE", -1, -1, -1) *# применяется для кнопок без ответа*class DatePicker:  
 months = ["Янв", "Фев", "Мар", "Апр", "Май", "Июн", "Июл", "Авг", "Сен", "Окт", "Ноя", "Дек"]  
  
 def \_\_init\_\_(self, year: int = datetime.now().year, month: int = datetime.now().month):  
 self.year = year  
 self.month = month  
  
 async def start\_picker(  
 self,  
 year: int = datetime.now().year - 20 *# 20 для того чтобы выбор года начинался для человека от 18 лет* ) -> InlineKeyboardMarkup:  
 inline\_kb = InlineKeyboardMarkup(row\_width=6)  
 *# first row - years* inline\_kb.row()  
 for value in range(year - 3, year + 3):  
 inline\_kb.insert(InlineKeyboardButton(  
 str(value),  
 callback\_data=date\_picker\_callback.new("SET-YEAR", value, -1, -1)  
 ))  
 *# nav buttons* inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 '<<',  
 callback\_data=date\_picker\_callback.new("PREV-YEARS", year, -1, -1)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '>>',  
 callback\_data=date\_picker\_callback.new("NEXT-YEARS", year, -1, -1)  
 ))  
  
 return inline\_kb  
  
 async def \_get\_month\_kb(self, year: int) -> InlineKeyboardMarkup:  
 inline\_kb = InlineKeyboardMarkup(row\_width=6)  
 *# первый ряд это года* inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(" ", callback\_data=ignore\_callback))  
 inline\_kb.insert(InlineKeyboardButton(  
 str(year),  
 callback\_data=date\_picker\_callback.new("START", year, -1, -1)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(" ", callback\_data=ignore\_callback))  
 *# два ряда кнопок по 6 месяцев* inline\_kb.row()  
 for month in self.months[0:6]:  
 inline\_kb.insert(InlineKeyboardButton(  
 month,  
 callback\_data=date\_picker\_callback.new("SET-MONTH", year, self.months.index(month) + 1, -1)  
 ))  
 inline\_kb.row()  
 for month in self.months[6:12]:  
 inline\_kb.insert(InlineKeyboardButton(  
 month,  
 callback\_data=date\_picker\_callback.new("SET-MONTH", year, self.months.index(month) + 1, -1)  
 ))  
 return inline\_kb  
  
 async def \_get\_days\_kb(self, year: int, month: int):  
 inline\_kb = InlineKeyboardMarkup(row\_width=7)  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 str(year),  
 callback\_data=date\_picker\_callback.new("START", year, -1, -1)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 self.months[month - 1],  
 callback\_data=date\_picker\_callback.new("SET-YEAR", year, -1, -1)  
 ))  
 inline\_kb.row()  
 for day in ["Пн", "Вт", "Ср", "Чт", "Пт", "Сб", "Вс"]:  
 inline\_kb.insert(InlineKeyboardButton(day, callback\_data=ignore\_callback))  
  
 month\_calendar = calendar.monthcalendar(year, month)  
 for week in month\_calendar:  
 inline\_kb.row()  
 for day in week:  
 if day == 0:  
 inline\_kb.insert(InlineKeyboardButton(" ", callback\_data=ignore\_callback))  
 continue  
 inline\_kb.insert(InlineKeyboardButton(  
 str(day), callback\_data=date\_picker\_callback.new("SET-DAY", year, month, day)  
 ))  
 return inline\_kb  
  
 async def process\_selection(self, query: CallbackQuery, callback\_data: dict) -> tuple:  
 return\_data: tuple = (False, None)  
 if callback\_data['act'] == "IGNORE":  
 await query.answer(cache\_time=60)  
 if callback\_data['act'] == "SET-YEAR":  
 await query.message.edit\_reply\_markup(await self.\_get\_month\_kb(int(callback\_data['year'])))  
 if callback\_data['act'] == "PREV-YEARS":  
 new\_year = int(callback\_data['year']) - 5  
 await query.message.edit\_reply\_markup(await self.start\_picker(new\_year))  
 if callback\_data['act'] == "NEXT-YEARS":  
 new\_year = int(callback\_data['year']) + 5  
 await query.message.edit\_reply\_markup(await self.start\_picker(new\_year))  
 if callback\_data['act'] == "START":  
 await query.message.edit\_reply\_markup(await self.start\_picker(int(callback\_data['year'])))  
 if callback\_data['act'] == "SET-MONTH":  
 await query.message.edit\_reply\_markup(  
 await self.\_get\_days\_kb(int(callback\_data['year']), int(callback\_data['month'])))  
 if callback\_data['act'] == "SET-DAY":  
 await query.message.delete\_reply\_markup() *# удаление инлайн-клавиатуры* return\_data = True, datetime(int(callback\_data['year']), int(callback\_data['month']),  
 int(callback\_data['day']))  
 return return\_data

**bot/keyboards/birthday\_slider.py**

from aiogram.types import \*  
from aiogram.utils.callback\_data import \*  
  
from bot.misc.formatting import format\_birthday  
  
slider\_callback = CallbackData('slider', 'action', 'days')  
  
  
class BirthdaySlider:  
 def \_\_init\_\_(self, birthdays):  
 self.birthdays = birthdays  
  
 async def get\_slider\_markup(self, day\_ind: int = 0) -> InlineKeyboardMarkup:  
 months = ["Января", "Февраля", "Марта", "Апреля", "Мая", "Июня",  
 "Июля", "Августа", "Сентября", "Октября", "Ноября", "Декабря"]  
 month, day = str.split(self.birthdays[day\_ind][1], '-')  
 inline\_kb = InlineKeyboardMarkup(row\_width=3)  
 inline\_kb.row()  
 inline\_kb.insert(InlineKeyboardButton(  
 '<<',  
 callback\_data=slider\_callback.new("PREV", (day\_ind - 1))  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '{} {}'.format(day, months[int(month) - 1]),  
 callback\_data=slider\_callback.new("IGNORE", day\_ind)  
 ))  
 inline\_kb.insert(InlineKeyboardButton(  
 '>>',  
 callback\_data=slider\_callback.new("NEXT", (day\_ind + 1))  
 ))  
 return inline\_kb  
  
 async def new\_text(self, day: int) -> str:  
 return await format\_birthday(self.birthdays, day)  
  
 async def selection(self, query: CallbackQuery, callback\_data: dict):  
 if callback\_data['action'] == "IGNORE":  
 await query.answer(cache\_time=60)  
 if callback\_data['action'] == "PREV":  
 if int(callback\_data['days']) >= 0:  
 await query.message.edit\_text(await self.new\_text(int(callback\_data['days'])), 'MarkdownV2',  
 reply\_markup=await self.get\_slider\_markup(int(callback\_data['days'])))  
 if callback\_data['action'] == "NEXT":  
 if int(callback\_data['days']) <= 13:  
 await query.message.edit\_text(await self.new\_text(int(callback\_data['days'])), 'MarkdownV2',  
 reply\_markup=await self.get\_slider\_markup(int(callback\_data['days'])))

**bot/keyboards/\_\_init\_\_.py**

from .inline import \*  
from .reply import \*

**bot/handlers/main.py**

from aiogram import Dispatcher  
  
from bot.handlers.admin.main import register\_admin\_handlers  
from bot.handlers.user.main import register\_user\_handlers  
from bot.handlers.registration import register\_registration\_handlers  
from bot.handlers.user.plus\_user import register\_user\_plus\_handlers  
  
  
def register\_all\_handlers(dp: Dispatcher) -> None:  
 handlers = (  
 register\_user\_plus\_handlers,  
 register\_user\_handlers,  
 register\_registration\_handlers,  
 register\_admin\_handlers  
 *# register\_other\_handlers,* )  
 for handler in handlers:  
 handler(dp)

**bot/handlers/\_\_init\_\_.py**

from .main import register\_all\_handlers

**bot/handlers/user/plus\_user.py**

from aiogram import Bot  
from aiogram.dispatcher import filters  
from aiogram.types import ParseMode  
  
from bot.database import database as db  
from bot.filters.main import \*  
from bot.keyboards import birthday\_slider  
from bot.keyboards.birthday\_slider import BirthdaySlider  
from bot.misc.formatting import format\_birthday, format\_founded\_users, format\_birthday\_by\_fio  
import bot.keyboards as kb  
from bot.misc.states import AdminStates as AdSt  
  
slider: BirthdaySlider  
  
  
async def birthday\_slider\_start(message: Message):  
 bot: Bot = message.bot  
 birthdays = await db.get\_users\_birthday(14)  
 text: str = await format\_birthday(birthdays, 0)  
 global slider  
 slider = BirthdaySlider(birthdays)  
 await bot.send\_message(message.chat.id, text, 'MarkdownV2', reply\_markup=await slider.get\_slider\_markup())  
  
  
async def birthday\_slider\_callback(callback\_query: CallbackQuery, callback\_data: dict, state: FSMContext):  
 global slider  
 await slider.selection(callback\_query, callback\_data)  
  
  
async def birthday\_find\_by\_fio(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id,  
 'Введите ФИО пользователя, если не знаете Фамилию,'  
 ' то поиск можно вести по имени или отчеству,\n '  
 '<b>{Нет}[пробел]{Имя}</b> и <b>{Нет}[пробел]{Нет}[пробел]{Отчество}</b>, соответственно',  
 ParseMode.HTML)  
 await state.set\_state(AdSt.INSERT\_USER\_FIO\_BIRTHDAY)  
  
  
async def birthday\_find\_by\_fio\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 users = db.find\_user(message.text.split())  
 founded = await format\_founded\_users(users, '/get\\_birthday\\_by\\_id')  
 founded\_users\_dict = founded[1]  
 await state.set\_state(AdSt.GET\_BIRTHDAY\_FIO)  
 bot\_me = await bot.send\_message(message.from\_user.id, founded[0], ParseMode.MARKDOWN\_V2)  
  
 async with state.proxy() as data:  
 data['list\_users\_msg'] = bot\_me  
 if len(founded\_users\_dict) == 0:  
 await state.finish()  
  
  
async def birthday\_find\_link\_callback(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 user\_id = message.text[19:]  
 birthday\_user = await db.get\_users\_birthday\_id(user\_id)  
 text = await format\_birthday\_by\_fio(birthday\_user)  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_msg']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 await bot.send\_message(message.chat.id, text, ParseMode.MARKDOWN\_V2)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
  
  
async def off\_birthday\_notif(message: Message, state: FSMContext):  
 await db.any\_command("UPDATE users SET birthday\_notif = false WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления отключены', reply\_markup=await kb.birthday\_menu(  
 db.get\_user\_birthday\_notif\_on(message.from\_user.id)[0]))  
  
  
async def on\_birthday\_notif(message: Message, state: FSMContext):  
 await db.any\_command("UPDATE users SET birthday\_notif = true WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления включены', reply\_markup=await kb.birthday\_menu(  
 db.get\_user\_birthday\_notif\_on(message.from\_user.id)[0]))  
  
  
async def get\_menu(message: Message, state: FSMContext):  
 await state.finish()  
 await message.bot.send\_message(message.from\_user.id, 'Возврат в главное меню',  
 reply\_markup=await kb.plus\_user\_start\_menu())  
  
  
async def birthday\_menu(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id, 'Укажите пункт меню',  
 reply\_markup=await kb.birthday\_menu(db.get\_user\_birthday\_notif\_on(message.from\_user.id)[0]))  
  
async def mock(message: Message, state: FSMContext):  
 pass  
def register\_user\_plus\_handlers(dp: Dispatcher) -> None:  
 *# handlers* dp.register\_message\_handler(birthday\_slider\_start, IsPlusUserOrAdminOrOwner(), content\_types=['text'],  
 text='Показать ДР на ближайшие 14 дней')  
 dp.register\_message\_handler(birthday\_find\_by\_fio, IsPlusUserOrAdminOrOwner(), content\_types=['text'],  
 text='Узнать день рождения по ФИО')  
 dp.register\_message\_handler(off\_birthday\_notif, IsPlusUserOrAdminOrOwner(), content\_types=['text'],  
 text='Отключить уведомления о ДР')  
 dp.register\_message\_handler(on\_birthday\_notif, IsPlusUserOrAdminOrOwner(), content\_types=['text'],  
 text='Включить уведомления о ДР')  
 dp.register\_message\_handler(birthday\_find\_by\_fio\_input, IsPlusUserOrAdminOrOwner(), content\_types=['text'],  
 state=AdSt.INSERT\_USER\_FIO\_BIRTHDAY)  
 dp.register\_message\_handler(birthday\_find\_link\_callback, IsAdminOrOwner(),  
 filters.RegexpCommandsFilter(regexp\_commands=['get\_birthday\_by\_id([0-9]\*)']),  
 state=AdSt.GET\_BIRTHDAY\_FIO)  
 dp.register\_message\_handler(mock, IsPlusUserOnly(), commands=['main\_menu'])  
  
 dp.register\_message\_handler(birthday\_menu, IsPlusUserOrAdminOrOwner(), text='Дни рождения')  
  
 *# callbacks* dp.register\_callback\_query\_handler(birthday\_slider\_callback, IsPlusUserOrAdminOrOwner(),  
 birthday\_slider.slider\_callback.filter())

**bot/handlers/user/main.py**

import time  
  
from aiogram import Bot  
from aiogram.types import \*  
  
import bot.keyboards as kb  
import bot.res.strings as st  
from bot.database import database as db  
from bot.filters.main import \*  
from bot.keyboards.events\_slider import EventsSlider, event\_slider\_callback\_data  
from bot.misc.anti\_swearing import is\_swearing, censoring\_text  
from bot.misc.formatting import format\_event\_extended  
from bot.misc.states import RegisterUser as RegState  
  
  
async def watch\_events(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 list\_event = await db.get\_list\_event(False)  
  
 event\_slider\_user = EventsSlider(list\_event, message.from\_user.id)  
 await bot.send\_message(message.chat.id,  
 await format\_event\_extended(list\_event[0][1], list\_event[0][2], list\_event[0][3],  
 list\_event[0][4], list\_event[0][5], list\_event[0][6]),  
 ParseMode.HTML,  
 reply\_markup=await event\_slider\_user.get\_slider\_markup()  
 )  
  
  
async def event\_slider\_callback(callback\_query: CallbackQuery, callback\_data: dict, state: FSMContext):  
 subscribe, event\_id = await EventsSlider(await db.get\_list\_event(), callback\_query.from\_user.id).selection(  
 callback\_query, callback\_data)  
 await callback\_query.answer()  
 if subscribe:  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id, 'Будем рады Вас видеть)')  
  
  
async def off\_events\_notif(message: Message, state: FSMContext):  
 await db.any\_command("UPDATE users SET event\_notif = false WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления отключены', reply\_markup=await kb.event\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def on\_events\_notif(message: Message, state: FSMContext):  
 await db.any\_command("UPDATE users SET event\_notif = true WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления включены', reply\_markup=await kb.event\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def swear\_check(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 if await is\_swearing(message.text):  
 await bot.send\_message(message.chat.id,  
 "\*Не материтесь\! Иначе можете быть заблокированы\!\*"  
 "\nПользователь, [{0}](tg://user?id={1}),сказал:\n".format(  
 message.from\_user.full\_name, message.from\_user.id) + await censoring\_text(  
 message.text),  
 'MarkdownV2')  
 block = await db.any\_command\_get\_bool("SELECT block\_for\_swearing({})".format(message.from\_user.id))  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 if block:  
 until: datetime = datetime.date.today() + datetime.timedelta(days=7)  
 await bot.restrict\_chat\_member(message.chat.id, message.from\_user.id,  
 ChatPermissions(False, False, False, False, False, False, False, False,  
 False, False, False, False, False, False, False), None,  
 int(time.mktime(until.timetuple()))  
 )  
 *# чтобы убрать пользователя из бана Настройки группы -> разрешения -> Исключения*async def event\_menu(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id, 'Выберите пункт меню',  
 reply\_markup=await kb.event\_menu(db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def off\_events\_notif(message: Message):  
 await db.any\_command("UPDATE users SET event\_notif = false WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления отключены', reply\_markup=await kb.event\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def on\_events\_notif(message: Message):  
 await db.any\_command("UPDATE users SET event\_notif = true WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления включены', reply\_markup=await kb.event\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def start\_help(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.chat.id, st.registration\_help, ParseMode.HTML)  
 await state.set\_state(RegState.HELP\_QUESTION)  
  
  
async def help\_question\_input(message: Message, state: FSMContext):  
 if message.text == '/main\_menu':  
 await state.finish()  
 else:  
 bot: Bot = message.bot  
 await bot.send\_message(message.chat.id,  
 "Ваш вопрос принят.\nОжидайте, скоро наши администраторы свяжутся с вами",  
 ParseMode.HTML)  
 await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID, message.text, message\_thread\_id=Env.QUESTION\_THREAD\_ID,  
 reply\_markup=await kb.question\_answer(message.from\_user.id), parse\_mode=ParseMode.HTML)  
  
 await state.finish()  
  
  
async def get\_menu(message: Message, state: FSMContext):  
 await state.finish()  
 await message.bot.send\_message(message.from\_user.id, 'Возврат в главное меню',  
 reply\_markup=await kb.regular\_user\_start\_menu())  
async def mock(message: Message, state: FSMContext):  
 pass  
  
def register\_user\_handlers(dp: Dispatcher) -> None:  
 dp.register\_message\_handler(watch\_events, IsRegularUserOrPlusUser(), content\_types=['text'],  
 text='Ближайшие мероприятия')  
 dp.register\_message\_handler(off\_events\_notif, IsRegularUserOrPlusUser(), content\_types=['text'],  
 text='Отключить уведомления')  
 dp.register\_message\_handler(on\_events\_notif, IsRegularUserOrPlusUser(), content\_types=['text'],  
 text='Включить уведомления')  
 dp.register\_message\_handler(event\_menu, IsRegularUserOrPlusUser(), text='Мероприятия клуба')  
  
 dp.register\_message\_handler(swear\_check, IsNOTNotificationGroupMessage(),  
 chat\_type=[ChatType.SUPERGROUP, ChatType.GROUP], content\_types=['text'])  
 dp.register\_callback\_query\_handler(event\_slider\_callback, IsRegularUserOrPlusUser(),  
 event\_slider\_callback\_data.filter())  
 dp.register\_message\_handler(help\_question\_input, IsRegularUserOrPlusUser(), content\_types=[ContentType.TEXT],  
 state=RegState.HELP\_QUESTION)  
 dp.register\_message\_handler(start\_help, IsRegularUserOrPlusUser(), text='Получить помощь/Задать вопрос')  
 dp.register\_message\_handler(mock, IsRegularUserOnly(), commands=['main\_menu'])

**bot/handlers/registration/main.py**

import datetime  
import re  
  
from aiogram import Dispatcher, Bot  
from aiogram.dispatcher import FSMContext  
from aiogram.dispatcher.filters import \*  
from aiogram.types import \*  
  
import bot.keyboards as kb  
import bot.res.strings as st  
from bot.database import database as conn  
from bot.env import Env  
from bot.filters.main import IsNotRegistered  
from bot.keyboards.date\_picker import date\_picker\_callback, DatePicker  
from bot.misc.formatting import replace\_markdown\_marks  
from bot.misc.states import RegisterUser as RegState  
  
  
async def cmd\_start(message: Message):  
 bot: Bot = message.bot  
 await bot.send\_message(message.chat.id, st.start\_response, reply\_markup=await kb.start\_reg\_message())  
  
  
async def get\_menu(message: Message, state: FSMContext):  
 await message.bot.send\_message(message.from\_user.id, 'Меню открыто', reply\_markup=await kb.registration\_menu())  
  
  
async def start\_register(callback\_query: CallbackQuery, state: FSMContext):  
 bot: Bot = callback\_query.bot  
 if callback\_query.data == 'register\_partner':  
 await bot.send\_message(callback\_query.message.chat.id, st.registr\_attention\_partner)  
 async with state.proxy() as data:  
 data['is\_business'] = 'partner'  
 if callback\_query.data == 'register\_ads':  
 await bot.send\_message(callback\_query.message.chat.id, st.registr\_attention\_ads)  
 async with state.proxy() as data:  
 data['is\_business'] = 'ads'  
 if callback\_query.data == 'register':  
 await bot.send\_message(callback\_query.message.chat.id, st.registr\_attention)  
  
 await bot.delete\_message(callback\_query.from\_user.id, callback\_query.message.message\_id)  
 await bot.answer\_callback\_query(callback\_query.id)  
 async with state.proxy() as data:  
 data['user\_id'] = callback\_query.from\_user.id  
 *# await bot.delete\_message(chat\_id=callback\_query.from\_user.id, message\_id=callback\_query.message.message\_id)* await bot.send\_message(callback\_query.message.chat.id, st.input\_surname, reply\_markup=await kb.registration\_menu())  
 await state.set\_state(RegState.INSERT\_SURNAME)  
  
  
async def permanent\_menu(message: Message, state: FSMContext) -> bool:  
 bot: Bot = message.bot  
 if message.text == 'Допустил ошибку/Начать заново':  
 await cmd\_start(message)  
 await state.finish()  
 return True  
 if message.text == 'Получить помощь по регистрации':  
 await bot.send\_message(message.chat.id, st.registration\_help)  
 await state.set\_state(RegState.HELP\_QUESTION)  
 return True  
  
  
async def help\_question\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 if message.text != 'Получить помощь по регистрации' and message.text != 'Допустил ошибку/Начать заново':  
 await bot.send\_message(message.chat.id, st.registration\_help\_answer)  
 await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID, message.text, message\_thread\_id=Env.QUESTION\_THREAD\_ID,  
 reply\_markup=await kb.question\_answer(message.from\_user.id))  
  
 await state.finish()  
  
  
async def surname\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['surname'] = message.text  
 await bot.send\_message(message.chat.id, st.input\_name)  
 await state.set\_state(RegState.INSERT\_NAME)  
  
  
async def name\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['name'] = message.text  
 await bot.send\_message(message.chat.id, st.input\_patronymic)  
 await state.set\_state(RegState.INSERT\_PATRONYMIC)  
  
  
async def patronymic\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['patronymic'] = message.text  
 business\_type: str = ''  
 try:  
 business\_type = data['is\_business']  
 except KeyError:  
 pass  
  
 if business\_type == 'partner' or business\_type == 'ads':  
 await state.set\_state(RegState.INSERT\_ABOUT\_BUSINESS)  
 if business\_type == 'partner':  
 await bot.send\_message(message.chat.id, st.about\_partner\_info)  
 else:  
 await bot.send\_message(message.chat.id, st.about\_ads\_info)  
 else:  
 await bot.send\_message(message.chat.id, st.input\_birthday, reply\_markup=await DatePicker().start\_picker())  
 await state.set\_state(RegState.NEUTRAL)  
  
  
async def info\_about\_business(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 business\_type: str = ''  
 try:  
 business\_type = data['is\_business']  
 except KeyError:  
 pass  
 if business\_type == 'partner':  
 await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID,  
 '[{} {} {}](tg://user?id={}) хочет стать партнером\!\n'  
 'В качестве данных о себе как о партнёре указал:\n{}'.format(  
 data['surname'], data['name'], data['patronymic'], data['user\_id'],  
 await replace\_markdown\_marks(message.text)),  
 ParseMode.MARKDOWN\_V2, message\_thread\_id=Env.NEW\_PARTNER\_THREAD\_ID)  
 if business\_type == 'ads':  
 await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID,  
 '[{} {} {}](tg://user?id={}) хотел бы сотрудничать как рекламодатель\n'  
 'В качестве данных о себе как об организации указал:\n\*{}\*'.format(  
 data['surname'], data['name'], data['patronymic'], data['user\_id'],  
 await replace\_markdown\_marks(message.text)),  
 ParseMode.MARKDOWN\_V2, message\_thread\_id=Env.NEW\_ADS\_THREAD\_ID)  
 await bot.send\_message(message.from\_user.id, st.business\_info\_answer)  
 await state.finish()  
  
  
async def birthday\_input(callback\_query: CallbackQuery, callback\_data: dict, state: FSMContext):  
 if await permanent\_menu(callback\_query.message, state):  
 return  
 date: datetime  
 selected, date = await DatePicker().process\_selection(callback\_query, callback\_data)  
 if selected:  
 await callback\_query.message.answer(f'Вы выбрали: {date.strftime("%Y-%m-%d")}')  
 async with state.proxy() as data:  
 data['birthday'] = date  
 await state.set\_state(RegState.INSERT\_ABOUT)  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id, st.input\_about)  
  
  
async def about\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['about'] = message.text  
 await state.set\_state(RegState.INSERT\_CAR\_PHOTO)  
 await bot.send\_message(message.chat.id, st.input\_car\_photo)  
  
  
async def car\_photo\_input(message: Message, state: FSMContext):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['photo'] = message.photo[0].file\_id *# Т.к. все файлы хранятся в Telegramе, то найти их можно по file\_id* await message.answer('Фото загружено!')  
 await bot.send\_photo(message.chat.id, photo=InputFile('bot//res//number\_plate.png'),  
 caption=st.input\_car\_number\_plate, reply\_markup=await kb.car\_number\_plate\_non\_rus())  
 await state.set\_state(RegState.INSERT\_NUMBER\_PLATE)  
  
  
async def number\_plate\_input(message: Message, state: FSMContext):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 if (re.fullmatch(r'[АВЕКМНОРСТУХ]\d{3}[АВЕКМНОРСТУХ]{2}-\d{2,3}', message.text) is not None) or (  
 re.fullmatch(r'[ABEKMHOPCTYX]\d{3}[ABEKMHOPCTYX]{2}-\d{2,3}', message.text) is not None):  
 async with state.proxy() as data:  
 data['number\_plate'] = message.text  
 await state.set\_state(RegState.NEUTRAL)  
 await bot.send\_message(message.chat.id, st.input\_phone\_number, reply\_markup=await kb.get\_phone\_number())  
 else:  
 await message.answer(st.err\_input\_number\_plate)  
  
  
async def non\_russian\_plate(callback\_query: CallbackQuery, state: FSMContext):  
 if await permanent\_menu(callback\_query.message, state):  
 return  
 bot: Bot = callback\_query.bot  
 await bot.answer\_callback\_query(callback\_query.id)  
 await state.set\_state(RegState.INSERT\_NON\_RUS\_PLATE)  
 await bot.send\_message(callback\_query.from\_user.id, st.input\_non\_rus\_car\_number\_plate)  
  
  
async def non\_russian\_plate\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['number\_plate'] = message.text  
 await state.set\_state(RegState.NEUTRAL)  
 await bot.send\_message(message.chat.id, st.input\_phone\_number, reply\_markup=await kb.get\_phone\_number())  
  
  
async def phone\_number\_input(message: Message, state: FSMContext):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 if message.contact is not None:  
 async with state.proxy() as data:  
 data['phone\_number'] = message.contact.phone\_number  
 await bot.send\_message(message.chat.id, 'Ваш номер телефона записан!',  
 reply\_markup=await kb.registration\_menu())  
 await bot.send\_message(message.chat.id, st.partner\_choice, reply\_markup=await kb.partner\_choice())  
  
  
async def wanna\_be\_partner(callback\_query: CallbackQuery, state: FSMContext):  
 if await permanent\_menu(callback\_query.message, state):  
 return  
 bot: Bot = callback\_query.bot  
 await bot.answer\_callback\_query(callback\_query.id)  
 await state.set\_state(RegState.INSERT\_PARTNER\_BUSINESS)  
 await bot.send\_message(callback\_query.from\_user.id, st.about\_partner\_info)  
  
  
async def wanna\_be\_partner\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if await permanent\_menu(message, state):  
 return  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 data['partner'] = message.text  
 await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID,  
 'Новый участник хочет стать партнером\!\n Это [{} {} {}](tg://user?id={})\n'  
 'В качестве данных о себе как о партнёре указал:\n{}'.format(  
 data['surname'], data['name'], data['patronymic'], data['user\_id'],  
 await replace\_markdown\_marks(message.text)),  
 'MarkdownV2',  
 message\_thread\_id=Env.NEW\_PARTNER\_THREAD\_ID)  
 await state.set\_state(RegState.NEUTRAL)  
 await bot.send\_message(message.chat.id, st.end\_registration, reply\_markup=await kb.end\_registration())  
  
  
async def no\_partner(callback\_query: CallbackQuery, state: FSMContext):  
 if await permanent\_menu(callback\_query.message, state):  
 return  
 bot: Bot = callback\_query.bot  
 await bot.answer\_callback\_query(callback\_query.id)  
 async with state.proxy() as data:  
 data['partner'] = 'Нет'  
 await state.set\_state(RegState.NEUTRAL)  
 await bot.send\_message(callback\_query.from\_user.id, st.end\_registration, reply\_markup=await kb.end\_registration())  
  
  
async def end\_registration(callback\_query: CallbackQuery, state: FSMContext):  
 bot: Bot = callback\_query.bot  
 await bot.answer\_callback\_query(callback\_query.id)  
 await conn.add\_user(state)  
 async with state.proxy() as data:  
 await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID,  
 "Новый участник\! Это [{} {} {}](tg://user?id={})".format(  
 data['surname'],  
 data['name'],  
 data['patronymic'],  
 data['user\_id']),  
 ParseMode.MARKDOWN\_V2, message\_thread\_id=Env.NEW\_MEMBER\_THREAD\_ID)  
 await state.finish()  
 await bot.send\_message(callback\_query.from\_user.id, st.finish\_registration,  
 reply\_markup=await kb.regular\_user\_start\_menu())  
  
  
async def mock(message: Message, state: FSMContext):  
 pass  
  
  
def register\_registration\_handlers(dp: Dispatcher) -> None:  
 *# message handlers* dp.register\_message\_handler(cmd\_start, IsNotRegistered(), CommandStart())  
 dp.register\_message\_handler(surname\_input, IsNotRegistered(), content\_types=['text'], state=RegState.INSERT\_SURNAME)  
 dp.register\_message\_handler(name\_input, IsNotRegistered(), content\_types=['text'], state=RegState.INSERT\_NAME)  
 dp.register\_message\_handler(patronymic\_input, IsNotRegistered(), content\_types=['text'],  
 state=RegState.INSERT\_PATRONYMIC)  
 dp.register\_message\_handler(about\_input, IsNotRegistered(), content\_types=[ContentType.TEXT],  
 state=RegState.INSERT\_ABOUT)  
 dp.register\_message\_handler(car\_photo\_input, IsNotRegistered(), content\_types=[ContentType.PHOTO],  
 state=RegState.INSERT\_CAR\_PHOTO)  
 dp.register\_message\_handler(number\_plate\_input, IsNotRegistered(), content\_types=[ContentType.TEXT],  
 state=RegState.INSERT\_NUMBER\_PLATE)  
 dp.register\_message\_handler(non\_russian\_plate\_input, IsNotRegistered(), content\_types=[ContentType.TEXT],  
 state=RegState.INSERT\_NON\_RUS\_PLATE)  
 dp.register\_message\_handler(phone\_number\_input, IsNotRegistered(), content\_types=[ContentType.CONTACT],  
 state=RegState.NEUTRAL)  
 dp.register\_message\_handler(wanna\_be\_partner\_input, IsNotRegistered(), content\_types=[ContentType.TEXT],  
 state=RegState.INSERT\_PARTNER\_BUSINESS)  
 dp.register\_message\_handler(help\_question\_input, IsNotRegistered(), content\_types=[ContentType.TEXT],  
 state=RegState.HELP\_QUESTION)  
 dp.register\_message\_handler(info\_about\_business, IsNotRegistered(), content\_types=[ContentType.TEXT],  
 state=RegState.INSERT\_ABOUT\_BUSINESS)  
 dp.register\_message\_handler(mock, IsNotRegistered(), commands=['main\_menu'])  
  
 *# callbacks* dp.register\_callback\_query\_handler(non\_russian\_plate, IsNotRegistered(), lambda l: l.data == "non\_rus\_plate",  
 state=RegState.INSERT\_NUMBER\_PLATE)  
  
 dp.register\_callback\_query\_handler(start\_register, IsNotRegistered(),  
 lambda l: l.data == "register" or  
 l.data == "register\_partner"  
 or l.data == "register\_ads")  
  
 dp.register\_callback\_query\_handler(birthday\_input, IsNotRegistered(), date\_picker\_callback.filter(),  
 state=RegState.NEUTRAL)  
 dp.register\_callback\_query\_handler(wanna\_be\_partner, IsNotRegistered(), lambda l: l.data == "wanna\_be\_partner",  
 state=RegState.NEUTRAL)  
 dp.register\_callback\_query\_handler(no\_partner, IsNotRegistered(), lambda l: l.data == "no\_partner",  
 state=RegState.NEUTRAL)  
 dp.register\_callback\_query\_handler(end\_registration, IsNotRegistered(), lambda l: l.data == "end\_registration",  
 state=RegState.NEUTRAL)

**bot/handlers/admin/owner.py**

from aiogram import Bot  
from aiogram.dispatcher import filters  
from aiogram.types import \*  
import bot.keyboards as kb  
from bot.filters.main import \*  
from bot.database import database as db  
from bot.misc.formatting import format\_founded\_users  
from bot.misc.states import UpdatePermissions as UpPe  
  
founded\_users\_dict: dict  
  
  
async def choice\_new\_admin\_add(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await message.answer('Введите Фамилию Имя Отчество, того человека которого хотите назначить админом '  
 '(Вводить нужно именно в указанном выше порядке в случае если не известна,'  
 ' например , фамилия, то пишите: Нет Иван Иванов и т.п.)', ParseMode.HTML)  
 await state.set\_state(UpPe.INSERT\_ADMIN\_FIO)  
  
  
async def input\_new\_admin\_fio(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 users = db.find\_user(message.text.split())  
 founded = await format\_founded\_users(users, "/set\\_admin\\_by\\_id\\_")  
 global founded\_users\_dict  
 founded\_users\_dict = founded[1]  
 bot\_me = await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID, founded[0], ParseMode.MARKDOWN\_V2)  
 await state.set\_state(UpPe.CHOICE\_ADMIN)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 async with state.proxy() as data:  
 data['list\_users\_mes'] = bot\_me  
 if len(founded\_users\_dict) == 0:  
 await state.finish()  
  
  
async def add\_admin\_link\_callback(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 left, right = message.text.split('@')  
 user\_id = left[17:]  
 db.set\_new\_admin(user\_id)  
 global founded\_users\_dict  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_mes']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 await message.answer('Новый админ\,{}\, добавлен\!'.format(founded\_users\_dict.get(str(user\_id))),  
 ParseMode.MARKDOWN\_V2)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 await state.finish()  
 await bot.send\_message(user\_id, 'Поздравляю Вы были назначены администратором бота',  
 reply\_markup=await kb.admin\_menu())  
  
  
async def choice\_delete\_admin(message: Message, state: FSMContext):  
 global founded\_users\_dict  
 bot: Bot = message.bot  
 users = db.find\_admins()  
 founded = await format\_founded\_users(users, "/delete\\_admin\\_id")  
 founded\_users\_dict = founded[1]  
 bot\_me = await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID, founded[0], ParseMode.MARKDOWN\_V2)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 async with state.proxy() as data:  
 data['list\_users\_mes'] = bot\_me  
 if len(founded\_users\_dict) == 0:  
 await state.finish()  
  
  
async def delete\_admin\_link\_callback(message: Message, state: FSMContext):  
 global founded\_users\_dict  
 bot: Bot = message.bot  
 left, right = message.text.split('@')  
 user\_id = left[16:]  
 db.delete\_admin(user\_id)  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_mes']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 await message.answer('Админ\,{}\, удален\!'.format(founded\_users\_dict.get(str(user\_id))), ParseMode.MARKDOWN\_V2)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 await state.finish()  
 await bot.send\_message(user\_id, 'Теперь вы обычный пользователь',  
 reply\_markup=await kb.regular\_user\_start\_menu())  
  
  
def get\_owner\_handlers(dp: Dispatcher) -> None:  
 *# callbacks* dp.register\_message\_handler(choice\_new\_admin\_add, IsOwnerOnly(), IsNotificationGroupMessage(),  
 commands=["set\_new\_admin"])  
 dp.register\_message\_handler(input\_new\_admin\_fio, IsOwnerOnly(), IsNotificationGroupMessage(),  
 state=UpPe.INSERT\_ADMIN\_FIO)  
 dp.register\_message\_handler(add\_admin\_link\_callback, IsOwnerOnly(), IsNotificationGroupMessage(),  
 filters.RegexpCommandsFilter(regexp\_commands=['set\_admin\_by\_id\_([0-9]\*)']),  
 state=UpPe.CHOICE\_ADMIN)  
 dp.register\_message\_handler(choice\_delete\_admin, IsOwnerOnly(), IsNotificationGroupMessage(),  
 commands=['delete\_admin'])  
 dp.register\_message\_handler(delete\_admin\_link\_callback, IsOwnerOnly(), IsNotificationGroupMessage(),  
 filters.RegexpCommandsFilter(regexp\_commands=['delete\_admin\_id([0-9]\*)']))

**bot/handlers/admin/nofitication\_group.py**

from aiogram import Bot, Dispatcher  
from aiogram.dispatcher import FSMContext  
from aiogram.types import \*  
import bot.keyboards as kb  
from bot.env import Env  
  
from bot.filters.main import IsNotificationGroupCallback  
  
  
async def answer\_on\_user\_question(callback: CallbackQuery, callback\_data: dict, state: FSMContext):  
 bot: Bot = callback.bot  
 await callback.answer()  
 await bot.edit\_message\_text(  
 'Отвечен ✔\n [Пользователю](tg://user?id={})'.format(  
 callback\_data['user\_id']) + '\nСам вопрос:' + callback.message.text,  
 message\_id=callback.message.message\_id,  
 reply\_markup=await kb.question\_delete(), chat\_id=Env.NOTIFICATION\_SUPER\_GROUP\_ID, parse\_mode='MarkdownV2')  
  
  
async def delete\_user\_question(callback: CallbackQuery, state: FSMContext):  
 bot: Bot = callback.bot  
 await callback.answer()  
 await bot.delete\_message(chat\_id=Env.NOTIFICATION\_SUPER\_GROUP\_ID, message\_id=callback.message.message\_id)  
  
  
def get\_notification\_handlers(dp: Dispatcher) -> None:  
 *# callbacks* dp.register\_callback\_query\_handler(answer\_on\_user\_question, IsNotificationGroupCallback(),  
 kb.question\_answer\_callbackdata.filter())  
 dp.register\_callback\_query\_handler(delete\_user\_question, IsNotificationGroupCallback(),  
 lambda l: l.data == 'delete\_question')

**bot/handlers/admin/main.py**

import re  
  
from aiogram import Bot  
from aiogram.dispatcher import filters  
from aiogram.types import \*  
from aiogram.utils.callback\_data import CallbackData  
from aiogram.utils.exceptions import ChatNotFound  
  
from bot.database.methods.import\_in\_file import get\_all\_users\_info  
from bot.filters.main import \*  
from bot.handlers.admin.nofitication\_group import get\_notification\_handlers  
from bot.handlers.admin.owner import get\_owner\_handlers  
from bot.keyboards.events\_slider\_admin import EventsSliderAdmin, event\_slider\_admin\_callback  
from bot.keyboards.simple\_calendar import SimpleCalendar, simple\_calendar\_callback  
from bot.misc.anti\_swearing import is\_swearing  
from bot.misc.formatting import \*  
from bot.database import database as db  
import bot.keyboards as kb  
from bot.res.states import CreateEvent as CrEv  
from bot.res.states import EditEvent as EdEv  
from bot.res.states import UpdatePermissions as UpPe  
from bot.res.states import AdminStates as AdSt  
  
founded\_users\_dict: dict  
  
  
async def choice\_new\_plus\_user\_add(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await message.answer('Введите Фамилию Имя Отчество, того человека которого хотите назначить админом '  
 '(Вводить нужно именно в указанном выше порядке в случае если не известна,'  
 ' например , фамилия, то пишите: Нет Иван Иванов и т.п.)', ParseMode.HTML)  
 await state.set\_state(UpPe.INSERT\_USER\_PLUS\_FIO)  
  
  
async def input\_new\_user\_plus\_fio(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 users = db.find\_user(message.text.split())  
 global founded\_users\_dict  
 founded: str = await format\_founded\_users(users, '/set\\_new\\_plus\\_user\\_id')  
 founded\_users\_dict = founded[1]  
 bot\_me = await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID, founded[0], ParseMode.MARKDOWN\_V2)  
 await state.set\_state(UpPe.CHOICE\_USER\_PLUS)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 async with state.proxy() as data:  
 data['list\_users\_mes'] = bot\_me  
 if len(founded\_users\_dict) == 0:  
 await state.finish()  
  
  
async def delete\_plus\_user\_choice(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 users = db.find\_plus\_users()  
 global founded\_users\_dict  
 founded: str = await format\_founded\_users(users, '/delete\\_plus\\_user\\_id')  
 founded\_users\_dict = founded[1]  
 bot\_me = await bot.send\_message(Env.NOTIFICATION\_SUPER\_GROUP\_ID, founded[0], ParseMode.MARKDOWN\_V2)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 async with state.proxy() as data:  
 data['list\_users\_mes'] = bot\_me  
 if len(founded\_users\_dict) == 0:  
 await state.finish()  
  
  
async def delete\_plus\_user\_link\_callback(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 left, right = message.text.split('@')  
 user\_id = left[20:]  
 db.delete\_plus\_user(user\_id)  
 global founded\_users\_dict  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_mes']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 await message.answer('Пользователь\,{}\, был лишен расширенного функционала\!'.format(  
 founded\_users\_dict.get(str(user\_id))), ParseMode.MARKDOWN\_V2)  
 await bot.send\_message(user\_id, 'Вы были лишены расширенного функционала', ParseMode.HTML,  
 reply\_markup=await kb.regular\_user\_start\_menu())  
 await bot.delete\_message(message.chat.id, message.message\_id)  
  
  
async def add\_plus\_user\_link\_callback(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 left, right = message.text.split('@')  
 user\_id = left[21:]  
 db.set\_new\_plus\_user(user\_id)  
 global founded\_users\_dict  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_mes']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 await message.answer('Новый пользователь\,{}\, с расширенной функциональностью добавлен\!'.format(  
 founded\_users\_dict.get(str(user\_id))), parse\_mode=ParseMode.MARKDOWN\_V2)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 await bot.send\_message(user\_id, 'Вам был предоставлен расширенный функционал', ParseMode.HTML,  
 reply\_markup=await kb.plus\_user\_start\_menu())  
 await state.finish()  
  
  
async def start\_create\_event(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await state.set\_state(CrEv.INSERT\_TITLE)  
 await bot.send\_message(message.chat.id, 'Введите название грядущего мероприятия (заголовок)')  
  
  
async def event\_title\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await state.set\_state(CrEv.INSERT\_DESCRIPTION)  
 async with state.proxy() as data:  
 data['title'] = message.text  
 await bot.send\_message(message.chat.id, 'Введите подробное описание того, что будет на этом мероприятии')  
  
  
async def event\_description\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await state.set\_state(CrEv.INSERT\_LOCATION)  
 async with state.proxy() as data:  
 data['description'] = message.text  
 await bot.send\_message(message.chat.id, 'Опишите место проведения мероприятия')  
  
  
async def event\_location\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await state.set\_state(CrEv.INSERT\_DATA)  
 async with state.proxy() as data:  
 data['location'] = message.text  
 await bot.send\_message(message.chat.id, 'Выберите дату когда должно будет пройти мероприятие',  
 reply\_markup=await SimpleCalendar().start\_calendar())  
  
  
async def event\_date\_input(callback\_query: CallbackQuery, callback\_data: CallbackData, state: FSMContext):  
 bot: Bot = callback\_query.bot  
 date: datetime  
 selected, date = await SimpleCalendar().process\_selection(callback\_query, callback\_data)  
 if selected:  
 await callback\_query.message.answer(f'Вы выбрали: {date.strftime("%Y-%m-%d")}', parse\_mode=ParseMode.HTML)  
 async with state.proxy() as data:  
 data['date'] = date  
 await state.set\_state(CrEv.INSERT\_TIME)  
 await bot.send\_message(callback\_query.from\_user.id,  
 'Введите время начала проведения мероприятия в 24х часовом формате (Пример: 13:33)',  
 parse\_mode=ParseMode.HTML)  
  
  
async def event\_time\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 if re.fullmatch(r'[012]?[0-9]:[0-5][0-9]', message.text) is not None:  
 await message.answer('Время учтено!')  
 async with state.proxy() as data:  
 data['time'] = message.text  
 async with state.proxy() as data:  
 await bot.send\_message(message.chat.id,  
 await format\_event(data['title'], data['date'], data['time'], data['location'],  
 data['description']), parse\_mode=ParseMode.HTML)  
 await bot.send\_message(message.chat.id, 'Подтвердите правильность введенных данных',  
 reply\_markup=await kb.end\_create\_event())  
 await state.set\_state(CrEv.FINISH\_CREATE)  
 else:  
 await message.answer('Время введено некорректно!')  
  
  
async def correct\_event(callback\_query: CallbackQuery, state: FSMContext):  
 await db.add\_event(state)  
 await callback\_query.answer()  
 await callback\_query.message.answer('Отлично! Мероприятие создано!', parse\_mode=ParseMode.HTML)  
 await state.finish()  
  
  
async def restart\_create\_event(callback\_query: CallbackQuery, state: FSMContext):  
 await state.finish()  
 await callback\_query.answer()  
 await start\_create\_event(callback\_query.message, state)  
  
  
event\_slider: EventsSliderAdmin  
  
  
async def watch\_events(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 list\_event = await db.get\_list\_event(True)  
 global event\_slider  
 event\_slider = EventsSliderAdmin(list\_event)  
 await bot.send\_message(message.chat.id,  
 await format\_event\_extended(list\_event[0][1], list\_event[0][2], list\_event[0][3],  
 list\_event[0][4], list\_event[0][5], list\_event[0][6]),  
 ParseMode.HTML,  
 reply\_markup=await event\_slider.get\_slider\_markup()  
 )  
  
  
async def event\_slider\_callback(callback\_query: CallbackQuery, callback\_data: dict, state: FSMContext):  
 global event\_slider\_user  
 edit, event\_id = await event\_slider.selection(callback\_query, callback\_data)  
 await callback\_query.answer()  
 if edit:  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id, 'Что нужно отредактировать?',  
 reply\_markup=await kb.edit\_event())  
 async with state.proxy() as data:  
 data['edit\_event\_id'] = event\_id  
  
  
async def event\_edit\_location(callback\_query: CallbackQuery, state: FSMContext):  
 await callback\_query.answer()  
 await callback\_query.message.delete()  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id, 'Введите новую локацию')  
 await state.set\_state(EdEv.INSERT\_LOCATION)  
  
  
async def edit\_event\_input\_location(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 async with state.proxy() as data:  
 db.edit\_event(data['edit\_event\_id'], message.text)  
 await bot.send\_message(message.chat.id, 'Локация изменена')  
 await state.finish()  
  
  
async def event\_edit\_date\_time(callback\_query: CallbackQuery, state: FSMContext):  
 await callback\_query.answer()  
 await callback\_query.message.delete()  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id,  
 'Введите дату и время, в формате: 2023-05-24/13:45', ParseMode.HTML)  
 await state.set\_state(EdEv.INSERT\_DATA\_TIME)  
  
  
async def edit\_event\_input\_date\_time(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 try:  
 date, time = message.text.split('/')  
 if re.fullmatch(r'[012][0-9]:[0-5][0-9]', time) is not None:  
 try:  
 datetime.date.fromisoformat(date)  
 async with state.proxy() as data:  
 db.edit\_event(data['edit\_event\_id'], date=date, time=time)  
 await bot.send\_message(message.chat.id, 'Дата/время изменены')  
 await state.finish()  
 except ValueError:  
 await bot.send\_message(message.chat.id, 'Неправильно введена дата формат: YYYY-MM-DD', ParseMode.HTML)  
 else:  
 await bot.send\_message(message.chat.id, 'Неправильно введено время формат: hh:mm')  
 except ValueError:  
 await bot.send\_message(message.chat.id, 'Неправильно введены данные')  
  
  
async def event\_edit\_title\_desc(callback\_query: CallbackQuery, state: FSMContext):  
 await callback\_query.answer()  
 await callback\_query.message.delete()  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id,  
 'Введите название мероприятия и его описание, в формате: Название/Описание')  
 await state.set\_state(EdEv.INSERT\_TITLE\_DESCRIPTION)  
  
  
async def edit\_event\_input\_title\_desc(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 try:  
 title, description = message.text.split('/')  
 async with state.proxy() as data:  
 db.edit\_event(data['edit\_event\_id'], title=title, description=description)  
 await bot.send\_message(message.chat.id, 'Название и описание мероприятия изменены')  
 await state.finish()  
 except ValueError:  
 await bot.send\_message(message.chat.id, 'Данные введены некорректно')  
  
  
async def event\_delete(callback\_query: CallbackQuery, state: FSMContext):  
 await callback\_query.answer()  
 await callback\_query.message.delete()  
 async with state.proxy() as data:  
 db.delete\_event(data['edit\_event\_id'])  
 await callback\_query.bot.send\_message(callback\_query.from\_user.id, 'Событие удалено!', ParseMode.HTML)  
 await state.finish()  
  
  
async def event\_cancel\_edit(callback\_query: CallbackQuery, state: FSMContext):  
 await callback\_query.answer()  
 await callback\_query.message.delete()  
 await state.finish()  
  
  
async def get\_menu(message: Message, state: FSMContext):  
 await state.finish()  
 await message.bot.send\_message(message.from\_user.id, 'Возврат в главное меню', reply\_markup=await kb.admin\_menu())  
  
  
async def birthday\_menu(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id, 'Укажите пункт меню',  
 reply\_markup=await kb.birthday\_menu(db.get\_user\_birthday\_notif\_on(message.from\_user.id)[0]))  
  
  
async def event\_menu(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id, 'Укажите пункт меню',  
 reply\_markup=await kb.event\_admin\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def off\_events\_notif(message: Message, state: FSMContext):  
 await db.any\_command("UPDATE users SET event\_notif = false WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления отключены', reply\_markup=await kb.event\_admin\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def on\_events\_notif(message: Message, state: FSMContext):  
 await db.any\_command("UPDATE users SET event\_notif = true WHERE user\_id = {}".format(message.from\_user.id))  
 await message.bot.send\_message(message.chat.id, 'Уведомления включены', reply\_markup=await kb.event\_admin\_menu(  
 db.get\_one\_user\_event\_notif\_on(message.from\_user.id)[0]))  
  
  
async def broadcast(message: Message, state: FSMContext):  
 await message.answer('Напишите сообщение, которое будет доставлено всем авторизованным пользователям в лс от бота',  
 reply\_markup=ReplyKeyboardRemove())  
 await state.set\_state(AdSt.INSERT\_BROADCAST)  
  
  
async def broadcast\_input(message: Message, state: FSMContext):  
 if not message.text.startswith('/'):  
 if not await is\_swearing(message.text):  
 await message.answer('Сообщение принято')  
 await state.finish()  
 user\_ids = await db.any\_command("SELECT user\_id from users")  
 for user\_id in user\_ids:  
 try:  
 mes\_pin = await message.bot.send\_message(user\_id[0], "<b>Важное объявление!</b>\n" + message.text,  
 ParseMode.HTML)  
 await message.bot.pin\_chat\_message(user\_id[0], mes\_pin.message\_id, True)  
 except ChatNotFound:  
 pass  
 else:  
 await message.answer('Такие сообщения не допускаются к публикации!', ParseMode.HTML)  
 else:  
 await state.finish()  
  
  
async def users\_info\_menu(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id, "Выберите пункт меню", reply\_markup=await kb.users\_info\_menu())  
  
  
async def user\_info\_by\_number\_plate(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id,  
 'Введите гос.номер в формате А333АА-54 (код-регион), '  
 'но т.к. номера могут быть иностранные, то допускается свободный ввод',  
 ParseMode.HTML)  
 await state.set\_state(AdSt.INSERT\_USER\_PLATE)  
  
  
async def user\_info\_by\_number\_plate\_input(message: Message, state: FSMContext):  
 if message.text == '/main\_menu':  
 await state.finish()  
 else:  
 bot: Bot = message.bot  
 user = db.find\_user\_by\_car(message.text)  
 text, file\_id = await user\_info(user)  
 await bot.send\_photo(message.from\_user.id, file\_id, text,  
 parse\_mode=ParseMode.HTML) *# Тут описание найденного пользователя*async def user\_info\_by\_fio(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await bot.send\_message(message.from\_user.id,  
 'Введите ФИО пользователя, если не знаете Фамилию,'  
 ' то поиск можно вести по имени или отчеству,\n '  
 '<b>{Нет}[пробел]{Имя}</b> и <b>{Нет}[пробел]{Нет}[пробел]{Отчество}</b>, соответственно',  
 ParseMode.HTML)  
 if message.text == 'Редактировать пользователя':  
 await state.set\_state(AdSt.EDIT\_USER)  
 if message.text == "Найти пользователя по ФИО":  
 await state.set\_state(AdSt.INSERT\_USER\_FIO)  
  
  
async def user\_info\_by\_fio\_input(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 users = db.find\_user(message.text.split())  
 global founded\_users\_dict  
 *# sm = await state.get\_state()* if await state.get\_state() == 'AdminStates:EDIT\_USER':  
 founded = await format\_founded\_users(users, '/edit\\_user\\_info\\_id')  
 founded\_users\_dict = founded[1]  
 await state.set\_state(AdSt.CHOICE\_USER\_EDIT)  
 if await state.get\_state() == 'AdminStates:INSERT\_USER\_FIO':  
 founded = await format\_founded\_users(users, '/get\\_user\\_info\\_id')  
 founded\_users\_dict = founded[1]  
 await state.set\_state(AdSt.CHOICE\_USER\_INFO)  
 bot\_me = await bot.send\_message(message.from\_user.id, founded[0], ParseMode.MARKDOWN\_V2)  
  
 async with state.proxy() as data:  
 data['list\_users\_msg'] = bot\_me  
 if len(founded\_users\_dict) == 0:  
 await state.finish()  
  
  
async def edit\_user\_info\_by\_fio\_link\_choice(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 user\_id = message.text[18:]  
 user = await db.get\_user\_by\_id(user\_id)  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_msg']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 data['edit\_user\_id'] = user\_id  
 text, file\_id = await user\_info(user)  
 await bot.send\_photo(message.from\_user.id, file\_id, text,  
 parse\_mode=ParseMode.HTML) *# Тут описание найденного пользователя* await bot.send\_message(message.chat.id, "Выберите поле которое хотите отредактировать\n"  
 "1.Дата рождения\n"  
 "2.Номер телефона\n"  
 "3.Род деятельности\n"  
 "4.Информация о партнерстве\n"  
 "5.Гос.номер\n\n "  
 "Отправьте:\n{номер}[пробел]{значение поля} \n\n"  
 "Если Вы хотите изменить фото авто, просто отправьте фото"  
 , parse\_mode=ParseMode.HTML)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 await state.set\_state(AdSt.EDIT\_DATA\_USER)  
  
  
async def edit\_user\_data(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 user\_id: int  
 async with state.proxy() as data:  
 user\_id = int(data['edit\_user\_id'])  
 if len(message.photo) != 0:  
 await state.finish()  
 await db.any\_command("UPDATE car SET car\_photo\_file\_id = '{}'"  
 " WHERE id = (select car\_id from users where user\_id={})".format(message.photo[0].file\_id,  
 user\_id))  
 await message.answer('Фото машины изменено')  
 else:  
 if re.fullmatch(r'[1-5] .\*', message.text) is not None:  
 left, right = message.text.split(maxsplit=1)  
 match left:  
 case '1': *# Дата рождения* try:  
 datetime.date.fromisoformat(right)  
 await db.any\_command(  
 "UPDATE users SET birthday = '{}' WHERE user\_id = {}".format(right, user\_id))  
 await message.answer('Дата рождения изменена')  
  
 except ValueError:  
 await bot.send\_message(message.chat.id, 'Неправильно введена дата формат: YYYY-MM-DD',  
 ParseMode.HTML)  
 case '2': *# Номер телефона* await db.any\_command(  
 "UPDATE users SET phone\_number = '{}' WHERE user\_id = {}".format(right, user\_id))  
 await message.answer('Номер телефона изменен')  
  
 case '3': *# Род деятельности* await db.any\_command("UPDATE users SET about = '{}' WHERE user\_id = {}".format(right, user\_id))  
 await message.answer('Род деятельности исправлен')  
  
 case '4': *# Информация о партнерстве* await db.any\_command("UPDATE users SET partner = '{}' WHERE user\_id = {}".format(right, user\_id))  
 await message.answer('Информация о партнерстве исправлена')  
  
 case '5': *# Гос.номер* await db.any\_command("UPDATE car SET number\_plate = '{}'"  
 " WHERE id = "  
 "(select car\_id from users where user\_id={})".format(right, user\_id))  
 await message.answer('Гос.номер исправлен')  
 pass  
 await state.finish()  
 else:  
 await message.answer('Должна быть только одна из цифр выше')  
  
  
async def user\_info\_by\_fio\_link\_choice(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 user\_id = message.text[17:]  
 user = await db.get\_user\_by\_id(user\_id)  
 async with state.proxy() as data:  
 list\_users\_mes = data['list\_users\_msg']  
 await bot.delete\_message(list\_users\_mes.chat.id, list\_users\_mes.message\_id)  
 text, file\_id = await user\_info(user)  
 await bot.send\_photo(message.from\_user.id, file\_id, text, parse\_mode=ParseMode.HTML)  
 await bot.delete\_message(message.chat.id, message.message\_id)  
 await state.finish()  
  
  
async def all\_users\_info(message: Message, state: FSMContext):  
 bot: Bot = message.bot  
 await get\_all\_users\_info()  
 await bot.send\_document(message.chat.id, open(r'all\_users\_data.xlsx', 'rb'),  
 caption='В данном файле содержится вся нужная информация о пользователях'  
 , parse\_mode=ParseMode.HTML)  
  
  
async def mock(message: Message, state: FSMContext):  
 pass  
  
  
def register\_admin\_handlers(dp: Dispatcher) -> None:  
 dp.register\_message\_handler(delete\_plus\_user\_choice, IsAdminOrOwner(), IsNotificationGroupMessage(),  
 commands=['delete\_plus\_user'])  
 dp.register\_message\_handler(delete\_plus\_user\_link\_callback, IsAdminOrOwner(), IsNotificationGroupMessage(),  
 filters.RegexpCommandsFilter(regexp\_commands=['delete\_plus\_user\_id([0-9]\*)']))  
  
 dp.register\_message\_handler(choice\_new\_plus\_user\_add, IsAdminOrOwner(), IsNotificationGroupMessage(),  
 commands=['set\_new\_plus\_user'])  
 dp.register\_message\_handler(add\_plus\_user\_link\_callback, IsAdminOrOwner(), IsNotificationGroupMessage(),  
 filters.RegexpCommandsFilter(regexp\_commands=['set\_new\_plus\_user\_id([0-9]\*)']),  
 state=UpPe.CHOICE\_USER\_PLUS)  
 dp.register\_message\_handler(input\_new\_user\_plus\_fio, IsAdminOrOwner(), IsNotificationGroupMessage(),  
 state=UpPe.INSERT\_USER\_PLUS\_FIO)  
 dp.register\_message\_handler(broadcast\_input, IsAdminOrOwner(), state=AdSt.INSERT\_BROADCAST)  
 dp.register\_message\_handler(user\_info\_by\_fio\_input, IsAdminOrOwner(), state=AdSt.EDIT\_USER)  
 dp.register\_message\_handler(edit\_user\_data, IsAdminOrOwner(), state=AdSt.EDIT\_DATA\_USER,  
 content\_types=['text', 'photo'])  
 dp.register\_message\_handler(user\_info\_by\_fio\_input, IsAdminOrOwner(), state=AdSt.INSERT\_USER\_FIO)  
 dp.register\_message\_handler(user\_info\_by\_number\_plate\_input, IsAdminOrOwner(), state=AdSt.INSERT\_USER\_PLATE)  
 dp.register\_message\_handler(user\_info\_by\_fio\_link\_choice, IsAdminOrOwner(),  
 filters.RegexpCommandsFilter(regexp\_commands=['get\_user\_info\_id([0-9]\*)']),  
 state=AdSt.CHOICE\_USER\_INFO)  
 dp.register\_message\_handler(edit\_user\_info\_by\_fio\_link\_choice, IsAdminOrOwner(),  
 filters.RegexpCommandsFilter(regexp\_commands=['edit\_user\_info\_id([0-9]\*)']),  
 state=AdSt.CHOICE\_USER\_EDIT)  
 dp.register\_message\_handler(mock, IsAdminOrOwner(), commands=['main\_menu'])  
 *# handlers to create an event* dp.register\_message\_handler(event\_title\_input, IsAdminOrOwner(), content\_types=['text'], state=CrEv.INSERT\_TITLE)  
 dp.register\_message\_handler(event\_description\_input, IsAdminOrOwner(), content\_types=['text'],  
 state=CrEv.INSERT\_DESCRIPTION)  
 dp.register\_message\_handler(event\_location\_input, IsAdminOrOwner(), content\_types=['text'],  
 state=CrEv.INSERT\_LOCATION)  
  
 dp.register\_callback\_query\_handler(event\_date\_input, IsAdminOrOwner(), simple\_calendar\_callback.filter(),  
 state=CrEv.INSERT\_DATA)  
 dp.register\_message\_handler(event\_time\_input, IsAdminOrOwner(), content\_types=['text'], state=CrEv.INSERT\_TIME)  
 dp.register\_callback\_query\_handler(correct\_event, IsAdminOrOwner(), lambda l: l.data == "correct\_event",  
 state=CrEv.FINISH\_CREATE)  
 dp.register\_callback\_query\_handler(restart\_create\_event, IsAdminOrOwner(), lambda l: l.data == "recreate\_event",  
 state=CrEv.FINISH\_CREATE)  
  
 dp.register\_callback\_query\_handler(event\_slider\_callback, IsAdminOrOwner(), event\_slider\_admin\_callback.filter())  
  
 *# edit events handlers* dp.register\_callback\_query\_handler(event\_edit\_location, IsAdminOrOwner(), lambda l: l.data == "location\_edit")  
 dp.register\_message\_handler(edit\_event\_input\_location, IsAdminOrOwner(), content\_types=['text'],  
 state=EdEv.INSERT\_LOCATION)  
 dp.register\_callback\_query\_handler(event\_edit\_date\_time, IsAdminOrOwner(), lambda l: l.data == "data/time\_edit")  
 dp.register\_message\_handler(edit\_event\_input\_date\_time, IsAdminOrOwner(), content\_types=['text'],  
 state=EdEv.INSERT\_DATA\_TIME)  
 dp.register\_callback\_query\_handler(event\_edit\_title\_desc, IsAdminOrOwner(),  
 lambda l: l.data == "title\_and\_desc\_edit")  
 dp.register\_message\_handler(edit\_event\_input\_title\_desc, IsAdminOrOwner(), content\_types=['text'],  
 state=EdEv.INSERT\_TITLE\_DESCRIPTION)  
 dp.register\_callback\_query\_handler(event\_delete, IsAdminOrOwner(), lambda l: l.data == "delete\_event")  
 dp.register\_callback\_query\_handler(event\_cancel\_edit, IsAdminOrOwner(), lambda l: l.data == "cancel\_event")  
  
 *# основное меню* dp.register\_message\_handler(birthday\_menu, IsAdminOrOwner(), text='Дни рождения')  
 dp.register\_message\_handler(broadcast, IsAdminOrOwner(), text='Рассылка')  
 dp.register\_message\_handler(event\_menu, IsAdminOrOwner(), text='Мероприятия клуба')  
 dp.register\_message\_handler(users\_info\_menu, IsAdminOrOwner(), text='Информация о пользователях')  
 dp.register\_message\_handler(user\_info\_by\_fio, IsAdminOrOwner(), content\_types=['text'],  
 text='Редактировать пользователя')  
  
 *# подпункты меню* dp.register\_message\_handler(watch\_events, IsAdminOrOwner(), text='Ближайшие мероприятия')  
 dp.register\_message\_handler(start\_create\_event, IsAdminOrOwner(), text='Создать мероприятие')  
 dp.register\_message\_handler(off\_events\_notif, IsAdminOrOwner(), content\_types=['text'],  
 text='Отключить уведомления')  
 dp.register\_message\_handler(on\_events\_notif, IsAdminOrOwner(), content\_types=['text'], text='Включить уведомления')  
 dp.register\_message\_handler(user\_info\_by\_fio, IsAdminOrOwner(), content\_types=['text'],  
 text='Найти пользователя по ФИО')  
 dp.register\_message\_handler(user\_info\_by\_number\_plate, IsAdminOrOwner(), content\_types=['text'],  
 text='Найти пользователя по Гос.Номеру')  
 dp.register\_message\_handler(all\_users\_info, IsAdminOrOwner(), content\_types=['text'],  
 text='Получить данные о всех пользователях')  
  
 *# Для Др подпункты такие же как и для user plus, реализация в: bot/handlers/user/plus\_user.py  
 # other admin handlers* get\_notification\_handlers(dp)  
 get\_owner\_handlers(dp)

**bot/filters/main.py**

from aiogram import Dispatcher  
from aiogram.dispatcher.filters import Filter  
from aiogram.types import Message, CallbackQuery  
  
from bot.database.database import \*  
from bot.env import Env  
  
  
class IsAdminOrOwner(Filter):  
 key = "is\_admin"  
  
 async def check(self, message: Message) -> bool:  
 try:  
 return await is\_admin(message.from\_user.id) or await is\_owner(message.from\_user.id)  
 except TypeError:  
 pass  
  
  
class IsOwnerOnly(Filter):  
 key = "is\_owner"  
  
 async def check(self, message: Message) -> bool:  
 try:  
 return await is\_owner(message.from\_user.id)  
 except TypeError:  
 pass  
  
  
class IsPlusUserOrAdminOrOwner(Filter):  
 key = "is\_plus\_user"  
  
 async def check(self, message: Message) -> bool:  
 try:  
 return await is\_plus\_user(message.from\_user.id) or await is\_admin(message.from\_user.id) or await is\_owner(  
 message.from\_user.id)  
 except TypeError:  
 pass  
  
  
class IsPlusUserOnly(Filter):  
 key = "is\_plus\_user\_only"  
  
 async def check(self, message: Message) -> bool:  
 try:  
 return await is\_plus\_user(message.from\_user.id) and not await is\_admin(  
 message.from\_user.id) and not await is\_owner(  
 message.from\_user.id)  
 except TypeError:  
 pass  
  
  
class IsNotRegistered(Filter):  
 key = "is\_not\_registered"  
  
 async def check(self, message: Message) -> bool:  
 return not (await is\_registered(message.from\_user.id))  
  
  
class IsRegularUserOrPlusUser(Filter):  
 key = "is\_regular\_user\_and\_plus"  
  
 async def check(self, message: Message) -> bool:  
 return await IsRegularUserOnly().check(message) or await IsPlusUserOnly().check(message)  
  
  
class IsRegistered(Filter):  
 key = "is\_regular\_user"  
  
 async def check(self, message: Message) -> bool:  
 return await is\_registered(message.from\_user.id)  
  
  
class IsRegularUserOnly(Filter):  
 key = "is\_regular\_user\_only"  
  
 async def check(self, message: Message) -> bool:  
 try:  
 return (not (await is\_plus\_user(message.from\_user.id)  
 or await is\_admin(message.from\_user.id)  
 or await is\_owner(message.from\_user.id)))  
 except TypeError:  
 pass  
  
  
class IsNotificationGroupCallback(Filter):  
 key = "is\_notification\_group"  
  
 async def check(self, callback: CallbackQuery) -> bool:  
 if callback.message.chat.id == Env.NOTIFICATION\_SUPER\_GROUP\_ID:  
 return True  
 else:  
 return False  
  
  
class IsNOTNotificationGroupMessage(Filter):  
 key = "is\_notification\_group\_msg"  
  
 async def check(self, msg: Message) -> bool:  
 if msg.chat.id == Env.NOTIFICATION\_SUPER\_GROUP\_ID:  
 return False  
 else:  
 return True  
  
  
class IsNotificationGroupMessage(Filter):  
 key = "is\_notification\_group\_msg"  
  
 async def check(self, msg: Message) -> bool:  
 if msg.chat.id == Env.NOTIFICATION\_SUPER\_GROUP\_ID:  
 return True  
 else:  
 return False  
  
  
def register\_all\_filters(dp: Dispatcher):  
 filters = (  
 IsAdminOrOwner,  
 IsOwnerOnly,  
 IsPlusUserOrAdminOrOwner,  
 IsPlusUserOnly,  
 IsNotRegistered,  
 IsRegularUserOrPlusUser,  
 IsRegistered,  
 IsRegularUserOnly,  
 IsNotificationGroupCallback,  
 IsNotificationGroupMessage,  
 IsNOTNotificationGroupMessage  
  
 )  
 for filter1 in filters:  
 dp.bind\_filter(filter1)

**bot/database/database.py**

import datetime  
  
import psycopg2  
from aiogram.dispatcher import FSMContext  
from psycopg2.\_psycopg import \*  
  
conn: connection  
  
  
def start():  
 global conn  
 try:  
 *# пытаемся подключиться к базе данных* conn = psycopg2.connect('postgresql://postgres:1824@localhost:5432/BMW\_Club\_NSK')  
 conn.set\_session(autocommit=True)  
 except Exception as e:  
 *# в случае сбоя подключения будет выведено сообщение в STDOUT* print('Can`t establish connection to database:' + str(e))  
  
  
async def add\_user(state: FSMContext):  
 async with state.proxy() as data:  
 birthday: datetime.datetime = data['birthday']  
 sql\_insert = "select insert\_user\_data({},'{}','{}','{}','{}','{}','{}','{}','{}','{}')". \  
 format(data['user\_id'],  
 data['name'],  
 data['surname'],  
 data['patronymic'],  
 birthday.strftime('%Y-%m-%d'),  
 data['about'],  
 data['photo'],  
 data['number\_plate'],  
 data['phone\_number'],  
 data['partner'])  
 with conn.cursor() as cur:  
 cur.execute(sql\_insert)  
  
 conn.commit()  
  
  
async def is\_admin(user\_id\_par) -> bool:  
 with conn.cursor() as cur:  
 cur.execute('select is\_admin from users where user\_id ={}'.format(user\_id\_par))  
 return cur.fetchone()[0]  
  
  
async def is\_plus\_user(user\_id\_par) -> bool:  
 with conn.cursor() as cur:  
 cur.execute('select is\_plus\_user from users where user\_id ={}'.format(user\_id\_par))  
 return cur.fetchone()[0]  
  
  
async def is\_owner(user\_id\_par) -> bool:  
 with conn.cursor() as cur:  
 cur.execute('select is\_owner from users where user\_id ={}'.format(user\_id\_par))  
 return cur.fetchone()[0]  
  
  
async def is\_registered(user\_id\_par) -> bool:  
 with conn.cursor() as cur:  
 cur.execute('select exists(select user\_id from users where user\_id = {})'.format(user\_id\_par))  
 return cur.fetchone()[0]  
  
  
async def get\_user\_by\_id(user\_id):  
 with conn.cursor() as cur:  
 cur.execute('SELECT user\_id,surname, first\_name, patronymic, birthday, '  
 'phone\_number,about,partner,is\_admin, is\_plus\_user,'  
 ' c.number\_plate, c.car\_photo\_file\_id FROM users join car c on c.id = users.car\_id'  
 ' where user\_id = {}'.format(user\_id))  
 return cur.fetchone()  
  
  
async def get\_users\_birthday(days: int) -> list:  
 *"""Кол-во дней означает для скольких дней от сегодняшнего дня должно быть показано ДР  
  
 Значение 1 - означает только сегодняшнюю дату  
 """* with conn.cursor() as cur:  
 date = datetime.date.today()  
 arr\_users = []  
 for i in range(days):  
 cur.execute(  
 "SELECT user\_id,surname,first\_name,patronymic,cast(date\_part('year',age(birthday)) AS INTEGER)+1"  
 " from users where date\_part('day',birthday) = {} "  
 "and date\_part('month',birthday) = {}".format(  
 date.strftime('%d'), date.strftime('%m')))  
 users: tuple = (cur.fetchall(), date.strftime('%m-%d'))  
 date += datetime.timedelta(days=1)  
 arr\_users.append(users)  
 return arr\_users  
  
  
async def get\_users\_birthday\_id(user\_id) -> list:  
 with conn.cursor() as cur:  
 cur.execute(  
 "SELECT user\_id,surname,first\_name,patronymic,cast(date\_part('year',age(birthday)) AS INTEGER)+1,birthday"  
 " from users where user\_id = {}".format(  
 user\_id))  
 return cur.fetchall()  
  
  
def get\_users\_birthday\_notif\_on() -> tuple:  
 with conn.cursor() as cur:  
 cur.execute(  
 "SELECT user\_id from users where birthday\_notif = TRUE and (is\_plus\_user = TRUE or is\_admin = TRUE or is\_owner=TRUE)")  
 return cur.fetchall()  
  
  
def get\_user\_birthday\_notif\_on(user\_id):  
 with conn.cursor() as cur:  
 cur.execute(  
 "SELECT birthday\_notif from users where user\_id = {}".format(user\_id))  
 return cur.fetchall()  
  
  
def set\_new\_admin(user\_id):  
 with conn.cursor() as cur:  
 cur.execute("UPDATE users SET is\_admin = TRUE WHERE user\_id = {}".format(user\_id))  
  
  
def delete\_admin(user\_id):  
 with conn.cursor() as cur:  
 cur.execute("UPDATE users SET is\_admin = FALSE WHERE user\_id = {}".format(user\_id))  
  
  
def set\_new\_plus\_user(user\_id):  
 with conn.cursor() as cur:  
 cur.execute("UPDATE users SET is\_plus\_user = TRUE WHERE user\_id = {}".format(user\_id))  
  
  
def delete\_plus\_user(user\_id):  
 with conn.cursor() as cur:  
 cur.execute("UPDATE users SET is\_plus\_user = FALSE WHERE user\_id = {}".format(user\_id))  
  
  
def find\_user(data\_fio: list):  
 sql\_command: str = "select user\_id,surname,first\_name,patronymic from users where "  
 if data\_fio[0].lower() != 'нет':  
 sql\_command += "surname = '{}'".format(data\_fio[0])  
 if len(data\_fio) > 1:  
 if data\_fio[0].lower() != 'нет':  
 sql\_command += " and "  
 if data\_fio[1].lower() != 'нет':  
 sql\_command += "first\_name = '{}' ".format(data\_fio[1])  
 if len(data\_fio) > 2:  
 if data\_fio[1].lower() != 'нет' and data\_fio[0].lower() != 'нет':  
 sql\_command += " and "  
 sql\_command += "patronymic = '{}'".format(data\_fio[2])  
 with conn.cursor() as cur:  
 cur.execute(sql\_command)  
 return cur.fetchall()  
  
  
def find\_user\_by\_car(number\_plate: str):  
 with conn.cursor() as cur:  
 cur.execute('SELECT user\_id,surname, first\_name, patronymic, birthday, '  
 'phone\_number,about,partner,is\_admin, is\_plus\_user,'  
 ' c.number\_plate, c.car\_photo\_file\_id FROM users join car c on c.id = users.car\_id '  
 " WHERE number\_plate = '{}'".format(number\_plate))  
 return cur.fetchone()  
  
  
def find\_admins():  
 with conn.cursor() as cur:  
 cur.execute("SELECT user\_id,surname,first\_name,patronymic FROM users where is\_admin = TRUE")  
 return cur.fetchall()  
  
  
def find\_plus\_users():  
 with conn.cursor() as cur:  
 cur.execute("SELECT user\_id,surname,first\_name,patronymic FROM users where is\_plus\_user = TRUE")  
 return cur.fetchall()  
  
  
async def add\_event(state: FSMContext):  
 async with state.proxy() as data:  
 date: datetime.datetime = data['date']  
 sql\_insert = "INSERT INTO events(title, date, time, location, description) VALUES ('{}','{}','{}','{}','{}')". \  
 format(data['title'],  
 date.strftime('%Y-%m-%d'),  
 data['time'],  
 data['location'],  
 data['description'])  
 with conn.cursor() as cur:  
 cur.execute(sql\_insert)  
  
  
async def get\_list\_event(all: bool = False):  
 sql\_command: str = "WHERE date >=(SELECT CURRENT\_DATE)"  
 with conn.cursor() as cur:  
 if all:  
 cur.execute("SELECT \* from events ORDER BY date ASC ")  
 else:  
 cur.execute("SELECT \* from events {} ORDER BY date ASC ".format(sql\_command))  
 return cur.fetchall()  
  
  
async def get\_tomorrow\_event():  
 date = datetime.date.today() + datetime.timedelta(days=1)  
 with conn.cursor() as cur:  
 cur.execute("SELECT \* from events WHERE date = '{}'".format(date.strftime('%Y-%m-%d')))  
 return cur.fetchall()  
  
  
def delete\_event(event\_id):  
 with conn.cursor() as cur:  
 cur.execute("DELETE FROM events WHERE id ={}".format(event\_id))  
  
  
def edit\_event(event\_id, location=None, date=None, time=None, description=None, title=None):  
 sql\_text = "UPDATE events SET "  
 if location is not None:  
 sql\_text += "location = '{}'".format(location)  
 if date is not None:  
 sql\_text += "date = '{}',time = '{}'".format(date, time)  
 if title is not None:  
 sql\_text += "title = '{}',description = '{}'".format(title, description)  
 sql\_text += "WHERE id = {}".format(event\_id)  
 with conn.cursor() as cur:  
 cur.execute(sql\_text)  
  
  
def get\_users\_event\_notif\_on() -> tuple:  
 with conn.cursor() as cur:  
 cur.execute("SELECT user\_id from users where event\_notif = TRUE")  
 return cur.fetchall()  
  
  
def get\_one\_user\_event\_notif\_on(user\_id):  
 with conn.cursor() as cur:  
 cur.execute("SELECT event\_notif from users where user\_id = {}".format(user\_id))  
 return cur.fetchall()  
  
  
async def unsubscribe\_on\_event(user\_id, event\_id):  
 with conn.cursor() as cur:  
 cur.execute("SELECT unsubscribe\_on\_event({},{})".format(user\_id, event\_id))  
  
  
async def subscribe\_on\_event(user\_id, event\_id):  
 with conn.cursor() as cur:  
 cur.execute("SELECT subscribe\_on\_event({},{})".format(user\_id, event\_id))  
  
  
async def is\_subscribe\_on\_event(event\_id, user\_id) -> bool:  
 with conn.cursor() as cur:  
 cur.execute("SELECT is\_subscribe\_on\_event({},{})".format(user\_id, event\_id))  
 return (cur.fetchall())[0][0]  
  
  
async def any\_command(sql: str):  
 with conn.cursor() as cur:  
 cur.execute(sql)  
 try:  
 return cur.fetchall()  
 except ProgrammingError:  
 return None  
  
  
async def any\_command\_get\_bool(sql: str):  
 with conn.cursor() as cur:  
 cur.execute(sql)  
 try:  
 return (cur.fetchall())[0][0]  
 except ProgrammingError:  
 return None

**bot/database/methods/import\_in\_file.py**

from openpyxl.workbook import Workbook  
from openpyxl.worksheet.worksheet import Worksheet  
  
import bot.database.database as db  
  
  
async def get\_all\_users\_info():  
 records = await db.any\_command("SELECT user\_id,surname,first\_name,patronymic,"  
 "birthday,phone\_number,about,partner,is\_admin,"  
 "is\_plus\_user,c.number\_plate,c.car\_photo\_file\_id "  
 "FROM users join car c on c.id = users.car\_id")  
 columns = (  
 "id пользователя", "Фамилия", "Имя", "Отчество",  
 "День рождения", "Номер телефона", "Род деятельности",  
 "Информация о партнере", "админ", "пользователь+", "Гос.номер", "file\_id авто")  
 file\_name = 'all\_users\_data.xlsx'  
 workbook = Workbook()  
 workbook.save(file\_name)  
 workbook.create\_sheet('data', 0)  
 worksheet: Worksheet = workbook['data']  
 worksheet.column\_dimensions['A'].width = 20  
 worksheet.column\_dimensions['B'].width = 20  
 worksheet.column\_dimensions['C'].width = 20  
 worksheet.column\_dimensions['D'].width = 20  
 worksheet.column\_dimensions['E'].width = 15  
 worksheet.column\_dimensions['F'].width = 15  
 worksheet.column\_dimensions['G'].width = 40  
 worksheet.column\_dimensions['H'].width = 40  
 worksheet.column\_dimensions['I'].width = 10  
 worksheet.column\_dimensions['J'].width = 15  
 worksheet.column\_dimensions['K'].width = 10  
 worksheet.column\_dimensions['L'].width = 85  
 worksheet.append(columns)  
 for row in records:  
 worksheet.append(row)  
 workbook.save(file\_name)  
 workbook.close()

**bot/database/methods/database\_polling.py**

from aiogram import Bot  
from apscheduler.schedulers.asyncio import AsyncIOScheduler  
  
from bot.database import database  
from bot.misc.formatting import \*  
  
  
async def start\_process\_db\_polling(bot: Bot):  
 scheduler = AsyncIOScheduler(timezone='Asia/Novosibirsk')  
 scheduler.add\_job(send\_birthday\_notif, trigger='cron', hour=11, minute=0, start\_date=datetime.datetime.now(),  
 kwargs={'bot': bot})  
 scheduler.add\_job(send\_event\_notification, trigger='cron', hour=13, minute=0, start\_date=datetime.datetime.now(),  
 kwargs={'bot': bot})  
 scheduler.add\_job(reset\_swearing, 'interval', weeks=1)  
 scheduler.start()  
  
  
months = ["Января", "Февраля", "Марта", "Апреля", "Мая", "Июня",  
 "Июля", "Августа", "Сентября", "Октября", "Ноября", "Декабря"]  
  
  
async def reset\_swearing():  
 await database.any\_command("TRUNCATE swearing\_users")  
  
  
async def send\_birthday\_notif(bot: Bot):  
 birthdays = await database.get\_users\_birthday(1)  
 if len(birthdays[0][0]) == 0:  
 return  
 users = database.get\_users\_birthday\_notif\_on()  
 month, day = str.split(birthdays[0][1], '-')  
 str\_birthdays\_notif: str = 'Сегодня\, {} {} \,день рождения у:'.format(day, months[int(month) - 1])  
 for birthday in birthdays[0][0]: *# [номер дня][0 - пользователи , 1 - дата (месяц и день)]* str\_birthdays\_notif += '\n[{} {} {}](tg://user?id={})\, исполняется {} '.format(birthday[1],  
 birthday[2],  
 birthday[3],  
 birthday[0],  
 birthday[4])  
 for user in users:  
 await bot.send\_message(user[0], str\_birthdays\_notif, parse\_mode='MarkdownV2')  
  
  
async def send\_event\_notification(bot: Bot):  
 events = await database.get\_tomorrow\_event()  
 if len(events) == 0:  
 return  
 users = database.get\_users\_event\_notif\_on()  
 event\_notif\_str: str = "Напоминание о грядущем мероприятиях, уже завтра:\n"  
 for event in events:  
 event\_notif\_str += await format\_event\_extended(event[1], event[2], event[3], event[4], event[5], event[6])  
 event\_notif\_str += '\n------------\n'  
 for user in users:  
 await bot.send\_message(user[0], event\_notif\_str, parse\_mode='HTML')

**SQL-код функций базы данных**

create function public.insert\_user\_data(user\_id\_into bigint, first\_name\_into text, surname\_into text, patronymic\_into text, birthday\_into date, about\_into text, photo bigint, number\_plate\_into text, phone\_number\_into text, partner\_into text) returns void  
 language plpgsql  
as  
$$  
DECLARE  
 car\_id\_into bigint;  
BEGIN  
 INSERT INTO car(car\_photo\_file\_id, number\_plate) VALUES (photo, number\_plate\_into);  
 SELECT id INTO car\_id\_into from car where number\_plate = number\_plate\_into;  
 INSERT INTO users(user\_id, first\_name, surname, patronymic, car\_id, birthday, about, phone\_number, partner)  
 VALUES (user\_id\_into, first\_name\_into, surname\_into, patronymic\_into, car\_id\_into, birthday\_into, about\_into,  
 phone\_number\_into, partner\_into);  
end;  
$$;  
  
alter function public.insert\_user\_data(bigint, text, text, text, date, text, bigint, text, text, text) owner to postgres;  
  
create function public.insert\_user\_data(user\_id\_into bigint, first\_name\_into text, surname\_into text, patronymic\_into text, birthday\_into date, about\_into text, photo text, number\_plate\_into text, phone\_number\_into text, partner\_into text) returns void  
 language plpgsql  
as  
$$  
DECLARE  
 car\_id\_into bigint;  
BEGIN  
 INSERT INTO car(car\_photo\_file\_id, number\_plate) VALUES (photo, number\_plate\_into);  
 SELECT id INTO car\_id\_into from car where number\_plate = number\_plate\_into;  
 INSERT INTO users(user\_id, first\_name, surname, patronymic, car\_id, birthday, about, phone\_number, partner)  
 VALUES (user\_id\_into, first\_name\_into, surname\_into, patronymic\_into, car\_id\_into, birthday\_into, about\_into,  
 phone\_number\_into, partner\_into);  
end;  
$$;  
  
alter function public.insert\_user\_data(bigint, text, text, text, date, text, text, text, text, text) owner to postgres;  
  
create function public.subscribe\_on\_event(user\_id bigint, event\_id bigint) returns void  
 language plpgsql  
as  
$$  
BEGIN  
 UPDATE events  
 SET people\_amount = people\_amount + 1, users\_id[people\_amount] = user\_id  
 where events.id = event\_id;  
end;  
$$;  
  
alter function public.subscribe\_on\_event(bigint, bigint) owner to postgres;  
  
create function public.unsubscribe\_on\_event(user\_id bigint, event\_id bigint) returns void  
 language plpgsql  
as  
$$  
BEGIN  
 UPDATE events  
 SET people\_amount = people\_amount - 1, users\_id = array\_remove(users\_id,user\_id)  
 where events.id = event\_id;  
end;  
$$;  
  
alter function public.unsubscribe\_on\_event(bigint, bigint) owner to postgres;  
  
create function public.is\_subscribe\_on\_event(user\_id\_arg bigint, event\_id bigint) returns boolean  
 language plpgsql  
as  
$$  
BEGIN  
 IF (SELECT (events.users\_id && ARRAY [user\_id\_arg]) from events where id = event\_id) IS FALSE OR  
 (SELECT (events.users\_id && ARRAY [user\_id\_arg]) from events where id = event\_id) IS NULL  
 THEN  
 RETURN FALSE;  
 ELSE  
 RETURN TRUE;  
 END IF;  
  
END ;  
$$;  
  
alter function public.is\_subscribe\_on\_event(bigint, bigint) owner to postgres;  
  
create function public.block\_for\_swearing(user\_id\_arg bigint) returns boolean  
 language plpgsql  
as  
$$  
BEGIN  
 IF NOT exists(SELECT swearing from swearing\_users where user\_id = user\_id\_arg) THEN  
 INSERT INTO swearing\_users(user\_id, swearing) VALUES (user\_id\_arg, 0);  
 end if;  
 UPDATE swearing\_users SET swearing = swearing + 1 WHERE user\_id = user\_id\_arg;  
 IF (SELECT swearing FROM swearing\_users WHERE user\_id = user\_id\_arg) > 2 THEN  
 RETURN TRUE;  
 end if;  
 RETURN FALSE;  
  
end;  
$$;  
  
alter function public.block\_for\_swearing(bigint) owner to postgres;

ПРИЛОЖЕНИЕ Б. АКТ О ВНЕДРЕНИИ РЕЗУЛЬТАТОВ ВКРБ

