

Колледж космического машиностроения и технологий

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка бота для Telegram: главный модуль»

Выполнил студент

Булатников Илья Сергеевич

Группа П1-17

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Подпись)

\_\_\_\_\_\_\_\_\_\_\_ (Дата сдачи работы)

Королёв 2020 г.

Оглавление

[**Введение** 2](#_Toc43978276)

[Глава 1. Теоретическая часть 3](#_Toc43978277)

[**1.1.** **Описание предметной области** 3](#_Toc43978278)

[**1.1.** **Описание существующих разработок** 5](#_Toc43978279)

[Глава 2. Проектная часть 7](#_Toc43978280)

[**2.1. Диаграмма прецедентов** 7](#_Toc43978281)

[**2.2. Выбор инструментов** 8](#_Toc43978282)

[**2.5. Описание главного модуля** 8](#_Toc43978283)

[**2.6. Описание спецификаций к модулям** 8](#_Toc43978284)

[**2.7. Описание модулей** 9](#_Toc43978285)

[**2.8. Описание тестовых наборов модулей** 11](#_Toc43978286)

[**2.9. Описание применения средств отладки** 12](#_Toc43978287)

[**Глава 3. Эксплуатационная часть** 13](#_Toc43978288)

[**Аннотация** 13](#_Toc43978289)

[**3.1. Руководство оператора** 14](#_Toc43978290)

[**3.4. Минимальный состав аппаратных средств** 17](#_Toc43978291)

[**Заключение** 17](#_Toc43978292)

## **Введение**

Целью данного курсового проекта является создание чат-бота для мессенджера Telegram. Мессенджеры и социальные сети в наши дни являются неотъемлемой частью современного общества, а чат-боты позволяют облегчить рутинные дела в этой информационной среде и сэкономить немного времени её пользователям.

В 1 главе рассматривается теоретическая часть программы, для понимания специфики выбранной тематики.

Во 2 главе рассматривается проектная часть программы, в которой присутствует описание программы.

В 3 главе рассматривается эксплуатационная часть программы, в которой описывается, как в программе работать.

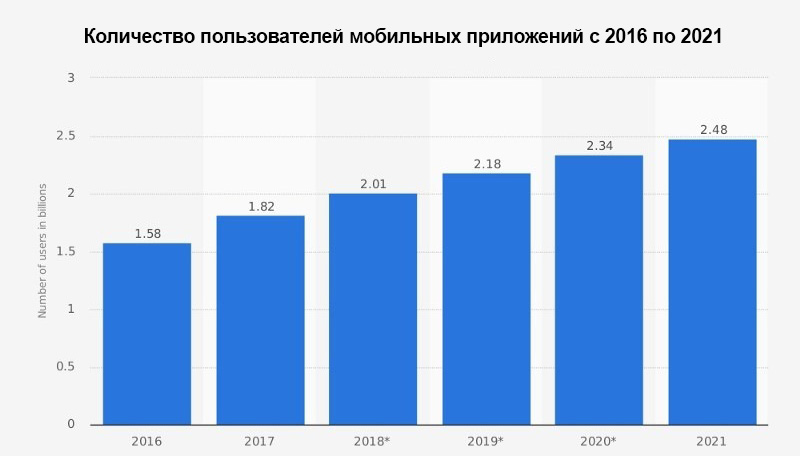
# Глава 1. Теоретическая часть

## **Описание предметной области**

В наши дни мессенджеры занимают весомую нишу на рынке мобильных приложений, это связано с ростом распространения смартфонов и доступного качественного беспроводного интернета.

Мессенджер ( Messenger) - это программа, мобильное приложение или веб-сервис для мгновенного обмена сообщениями.

Чаще всего под мессенджером понимают программу, в которую вы пишите сообщения и где вы их читаете. Однако, за каждой такой программой стоит сеть обмена сообщениями, которая тоже входит в понятие "мессенджер". Это может быть сеть внутри вашей компании, а может быть глобальная сеть, например Jabber.



**Рисунок 1.Диаграмма роста пользователей на мобильном рынке**

Нужно сказать, что понятие мессенджера уже давно не связывают только с обменом текстовыми сообщениями. Современные мессенджеры уже стали полноценными коммуникационными центрами, которые помимо обмена сообщениями реализуют голосовую и видеосвязь, обмен файлами, веб-конференции.

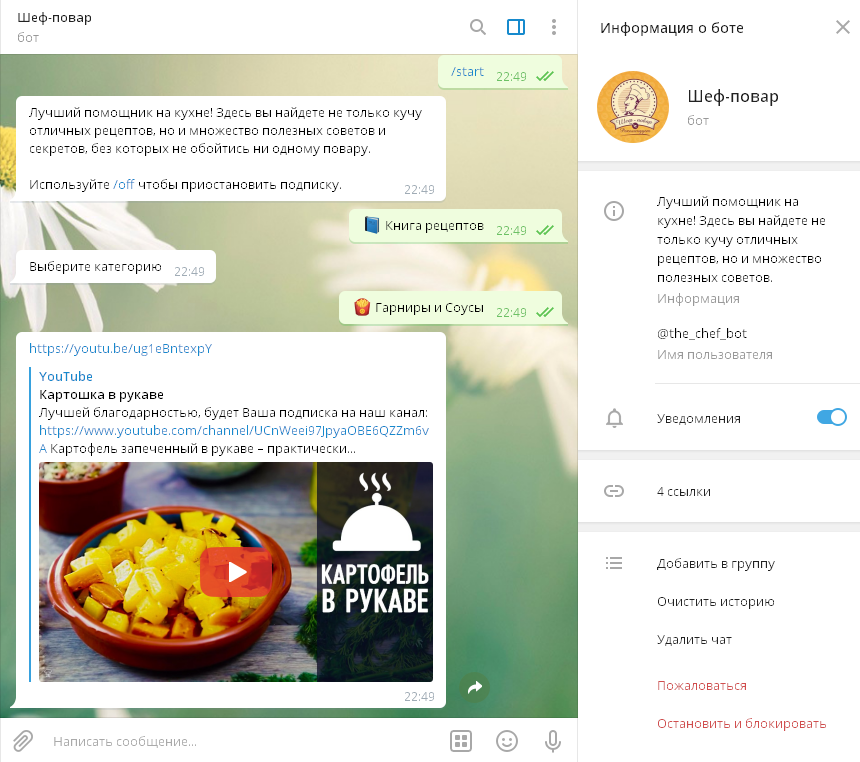
Наиболее популярные мессенджеры - это WhatsApp, Viber, Telegram, Facebook Messenger, Skype, ICQ.

Chat-bot (чат-бот) – это виртуальный собеседник для мессенджера с дополнительным функционалом.

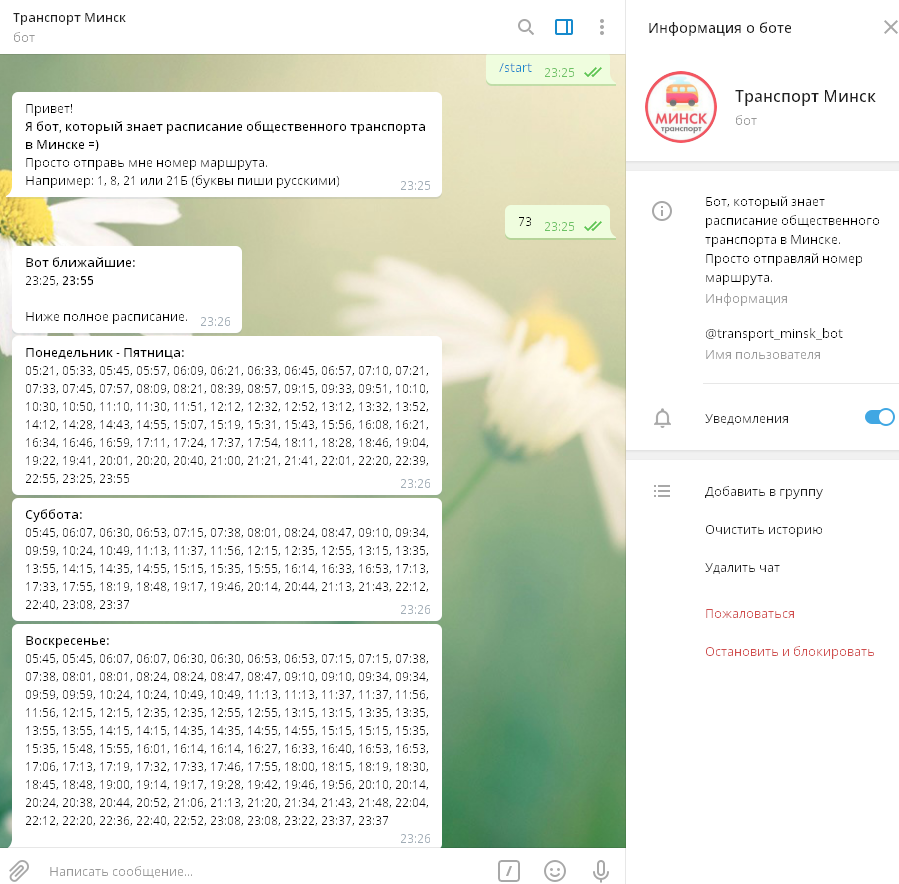
Такого бота можно использовать для рассылки сообщений или сбора информации.

## **Описание существующих разработок**

Сферы применения ботов, довольно разнообразны, например, есть бот Шеф-повар, он высылает рецепт на определенное или случайное блюдо из категории.



**Рисунок 2.Бот Шеф-повар**

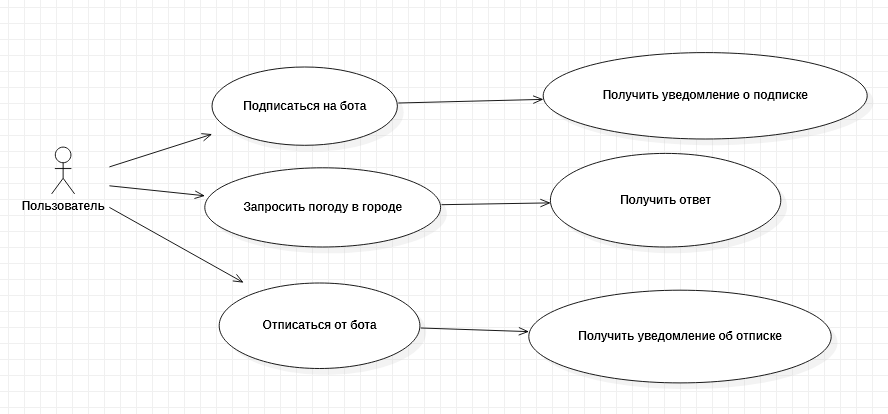
Ещё есть бот для просмотра расписание автобусов по городу, например в Минске: ******

**Рисунок 3.Бот для расписания автотранспорта**

# Глава 2. Проектная часть

## **2.1. Диаграмма прецедентов**

В этом разделе представлена диаграмма прецедентов. На диаграмме (рисунок№ 4) показаны возможные действия пользователя.



**Рисунок 4.Диаграмма "Пользователь"**

## **2.2. Выбор инструментов**

Мой выбор инструментов зависел от нескольких факторов:

* Наличие документации на русском языке.
* Скорость разработки
* Наличие API для написания Telegram-бота на данном языке
* Наличие бесплатной среды разработки

Основываясь на этих критериях был выбран язык программирования Python и Aiogram API.

Ссылки на скачивание:

* <https://www.python.org/ftp/python/3.8.3/python-3.8.3.exe>
* <https://docs.aiogram.dev/en/latest/install.html>

## **2.5. Описание главного модуля**

В главный модуль входит реализация команд для бота

В главный модуль импортируется 3 второстепенных модуля.

Также в главный модуль входит код, который запускает саму программу. Полный код главного модуля находится в приложении 1.

## **2.6. Описание спецификаций к модулям**

В данном разделе описаны спецификации к модулям. Спецификации к

модулям заключается в следующем: использование методов библиотеки aiogram и pyowm

Методы:

 await message.answer – отправка сообщения.

 start\_polling – запуск бота.

 weather\_manager() – объект класса погода.

 wind() – ветер.

 weather\_at\_place() – определение погоды в введенном городе.

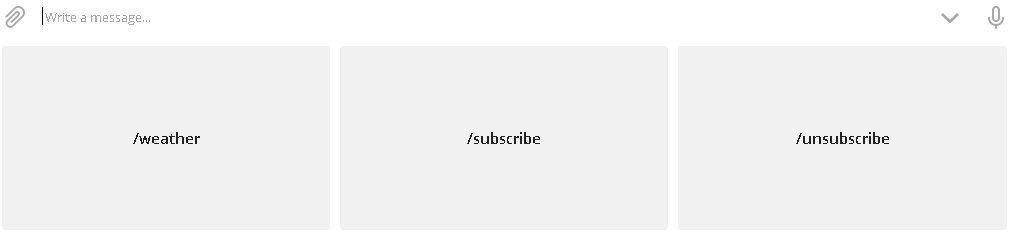
 humidity – влажность.

 temperature – температура.

 detailed\_status – описание погоды.

## **2.7. Описание модулей**

Рассмотрим один из модулей программы - это модуль экранная клавиатура



**Рисунок 5. Экранная клавиатура**

В данном окне мы можем выбрать что мы хотим сделать, подписаться/отписаться, запросить текущую погоду в введенном регионе.

**Листинг модуля Keybord.py**

from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup,\

ReplyKeyboardRemove, KeyboardButton, ReplyKeyboardMarkup  
  
button1 = InlineKeyboardButton(" ", callback\_data='btn1')  
button2 = InlineKeyboardButton(" ", callback\_data='btn2')  
button3 = InlineKeyboardButton(" ", callback\_data='btn3')  
button4 = InlineKeyboardButton(" ", callback\_data='btn4')  
button5 = InlineKeyboardButton(" ", callback\_data='btn5')  
button6 = InlineKeyboardButton(" ", callback\_data='btn6')  
button7 = InlineKeyboardButton(" ", callback\_data='btn7')  
button8 = InlineKeyboardButton(" ", callback\_data='btn8')  
button9 = InlineKeyboardButton(" ", callback\_data='btn9')  
  
button10 = KeyboardButton('/weather')  
button11 = KeyboardButton('/subscribe')  
button12 = KeyboardButton('/unsubscribe')  
  
markup3 = ReplyKeyboardMarkup().row(  
    button10, button11, button12  
)  
  
inline\_kb1 = InlineKeyboardMarkup()  
  
inline\_kb1.add(button1, button2, button3, button4, button5, button6, button7, button8, button9)

**Листинг модуля geo.py**

import geocoder  
import pyowm  
  
  
def weatherd(city):  
    owm = pyowm.OWM('a7a5f151b3845f1d0a5979f764dbb267')  
    mgr = owm.weather\_manager()      
    observation = mgr.weather\_at\_place(city + ',rus')  
    w = observation.weather  
    temp = w.temperature('celsius')["temp"]  
    wind = w.wind()["speed"]  
    hum = w.humidity  
  
    answer = "В городе " + city + " сейчас " + w.detailed\_status + "\n---------------------"  
    answer += "\nТемпература: " + str(temp) + "°C" + "\n---------------------" + "\nСкорость ветра: " + str(wind) + "м/с." + "\n---------------------" + "\nВлажность: " + str(hum) + "%" + "\n---------------------"  
    return answer

Листинг модуля sqliter.py

import sqlite3  
  
class SQLighter:  
  
    def \_\_init\_\_(self, database):  
        """Подключаемся к БД и сохраняем курсор соединения"""  
        self.connection = sqlite3.connect(database)  
        self.cursor = self.connection.cursor()  
  
    def get\_subscriptions(self, status = True):  
        """Получаем всех активных подписчиков бота"""  
        with self.connection:  
            return self.cursor.execute("SELECT \* FROM `subscriptions` WHERE `status` = ?", (status,)).fetchall()  
  
    def subscriber\_exists(self, user\_id):  
        """Проверяем, есть ли уже юзер в базе"""  
        with self.connection:  
            result = self.cursor.execute('SELECT \* FROM `subscriptions` WHERE `user\_id` = ?', (user\_id,)).fetchall()  
            return bool(len(result))  
  
    def add\_subscriber(self, user\_id, status = True):  
        """Добавляем нового подписчика"""  
        with self.connection:  
            return self.cursor.execute("INSERT INTO `subscriptions` (`user\_id`, `status`) VALUES(?,?)", (user\_id,status))  
  
    def update\_subscription(self, user\_id, status):  
        """Обновляем статус подписки пользователя"""  
        with self.connection:  
            return self.cursor.execute("UPDATE `subscriptions` SET `status` = ? WHERE `user\_id` = ?", (status, user\_id))  
  
    def close(self):  
        """Закрываем соединение с БД"""  
        self.connection.close()

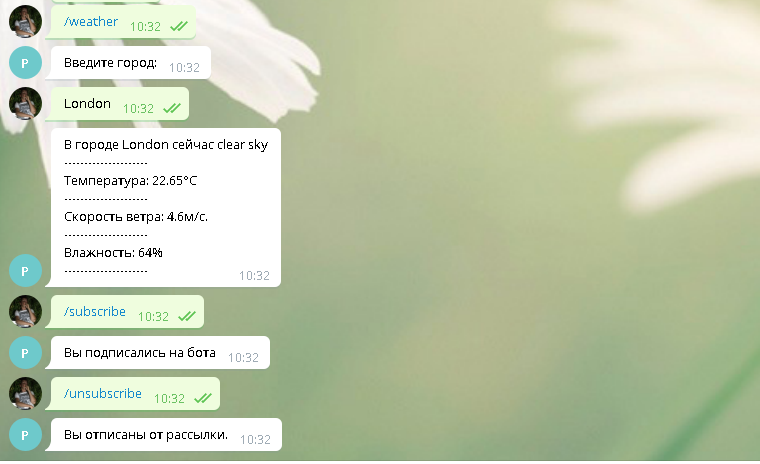
## **2.8. Описание тестовых наборов модулей**

В каждом модуле требуется вводить определенные значения. Там, где

имеется поле ввода следует ввести необходимое название населенного пункта.

Если, не соблюдать это правило, то программа не выполнит свою функцию.

Пример выпадающего списка можно увидеть на рисунке № 6.

****

**Рисунок 6. Пример работы бота**

## **2.9. Описание применения средств отладки**

При проведении тестов было выявлено:

Ошибок нет.

## **Глава 3. Эксплуатационная часть**

## **Аннотация**

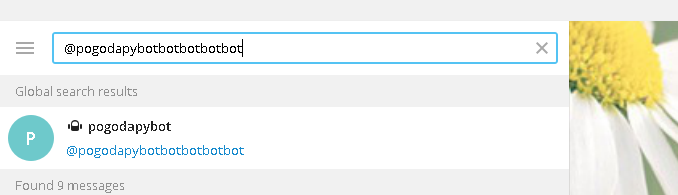
В данной главе приведено руководство оператора по применению и

использованию данного чат-бота в мессенджере Telegram, а также минимальный состав аппаратных средств для работы с приложение.

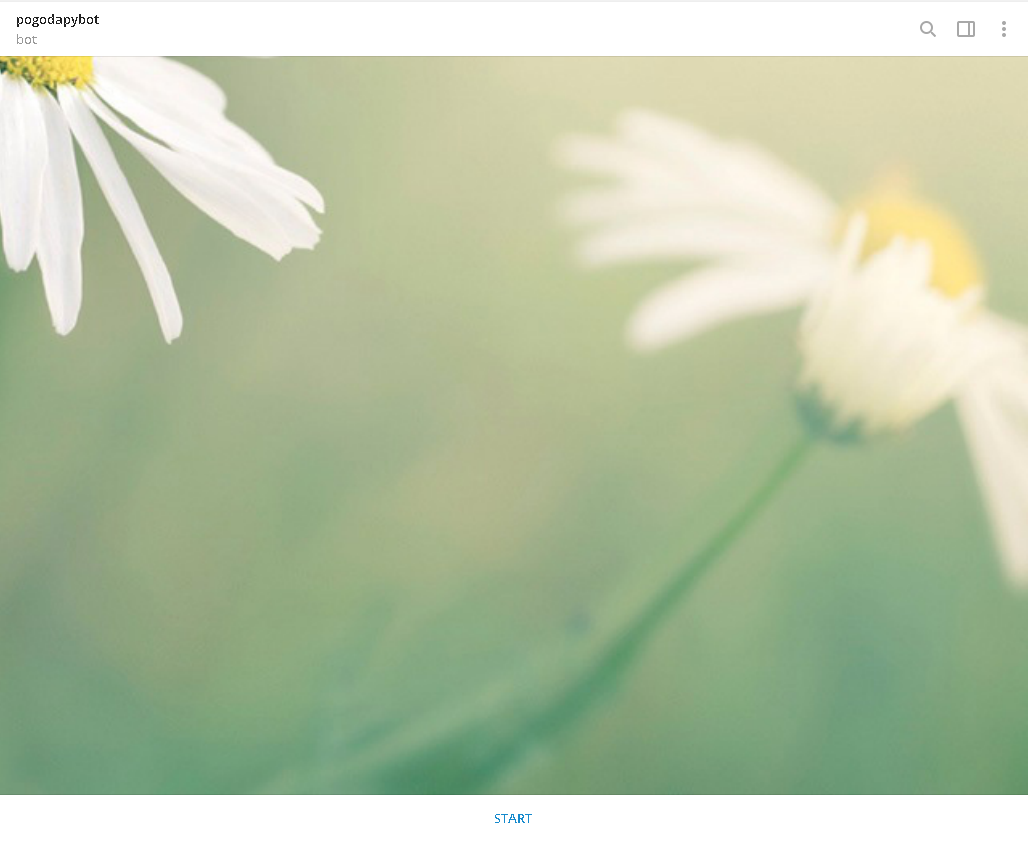
## **3.1. Руководство оператора**

Необходимо убедиться в наличии установленного приложения Telegram , в противном случае его установить с официального сайта разработчика (<https://telegram.org/apps>).

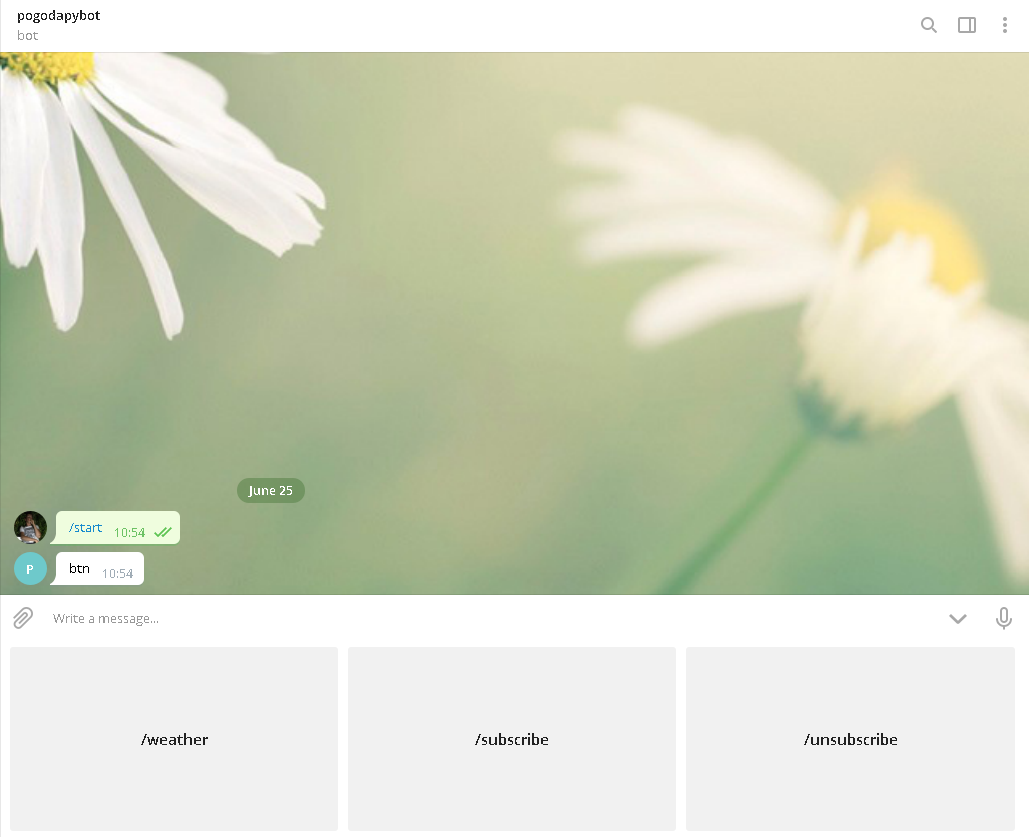
И после этого следовать инструкции приведенной ниже:

Шаг 1. Найти бота в поиске 

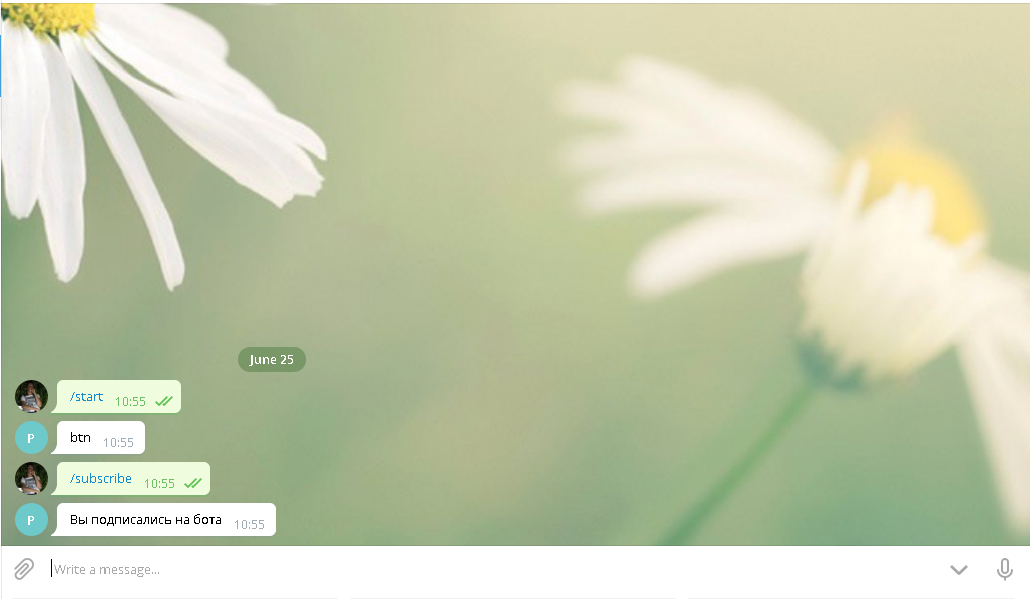
**Рисунок 7. Поиск чат-бота**

Шаг 2. Нажать кнопку START

**Рисунок 8. Начальное окно чата**

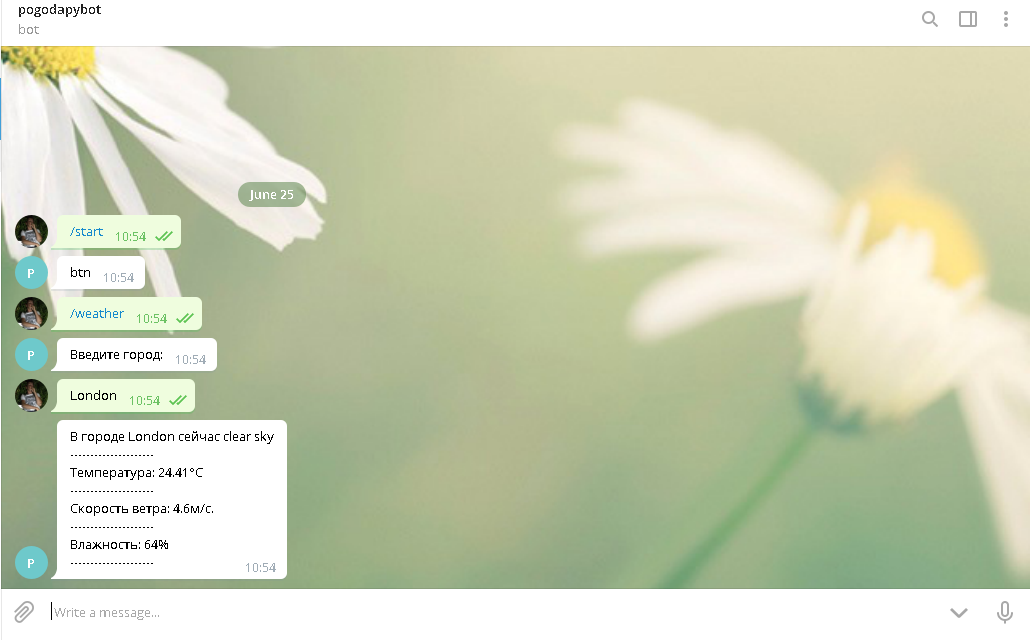
Шаг 3. На появившейся клавиатуре нажать кнопку /subscribe 

**Рисунок 9. Экранная клавиатура с доступными командами**



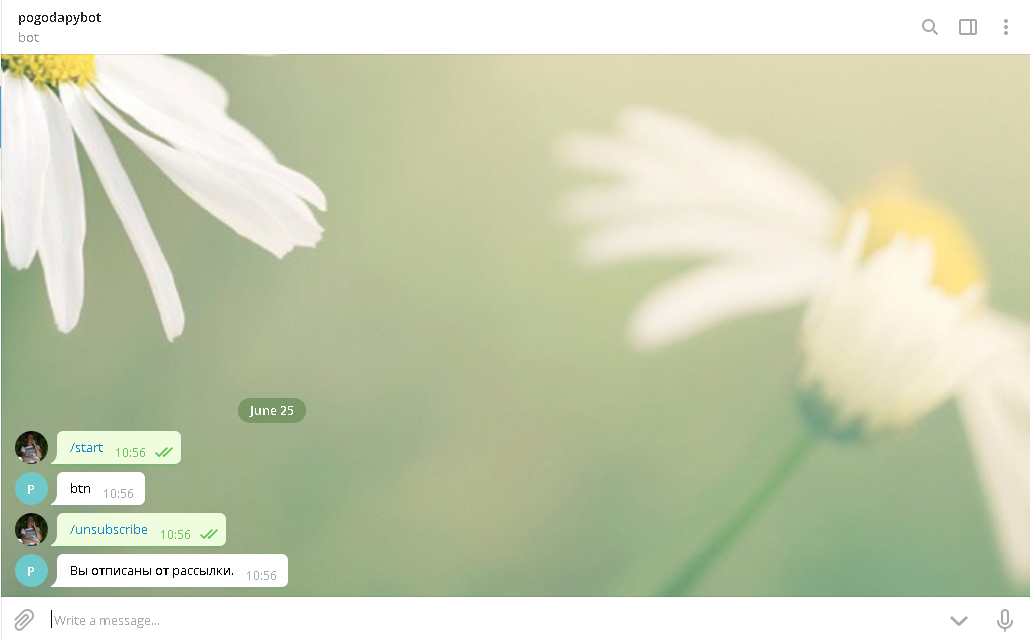
**Рисунок 10.Резульат работы /subscribe**

Шаг 4. Нажать кнопку /weather и затем ввести город



**Рисунок 11. Результат работы /weather**

Шаг 5. По желанию отписаться от бота (кнопка /unsubscribe)



**Рисунок 12. Резульать работы /unsubscribe**

## **3.4. Минимальный состав аппаратных средств**

Телефон или компьютер с доступом в интернет с установленной программой Telegram.

## **Заключение**

В данном проекте было сделано много работы, изучено довольно большое количество материала по разным сферам. В результате я расширил свои знания в языках программирования и создании БД.

# Список литературы и интернет-источников

API для главного модуля программы:

* <https://docs.aiogram.dev/en/latest/_modules/aiogram/bot/api.html>

API для модуля погоды:

* <https://pyowm.readthedocs.io/en/latest/>

Информация по работе с Telegram API:

* <https://core.telegram.org/>

API для модуля БД sqlite3:

* <https://docs.python.org/2/library/sqlite3.html>

# Приложение 1. Код главного меню.

#=========================

#Главный модуль программы

#=========================

import keybord as kb

import logging

import asyncio

import pyowm

import geo as geo

from datetime import datetime

from aiogram import Bot, Dispatcher, executor, types

import aiogram.types.base

from sqlighter import SQLighter

# задаем уровень логов

logging.basicConfig(level=logging.INFO)

# инициализируем бота

bot = Bot(token='1296584769:AAEu7y4Qjz9y7k6zA0rn8Tz9PmzUnJA4mB0')

dp = Dispatcher(bot)

# инициализируем соединение с БД

db = SQLighter('db.db')

# Команда активации подписки

@dp.message\_handler(commands=['subscribe'])

async def subscribe(message: types.Message):

if(not db.subscriber\_exists(message.from\_user.id)):

db.add\_subscriber(message.from\_user.id)

else:

db.update\_subscription(message.from\_user.id, True)

# Команда отписки

@dp.message\_handler(commands=['unsubscribe'])

async def unsubscribe(message: types.Message):

if(not db.subscriber\_exists(message.from\_user.id)):

db.add\_subscriber(message.from\_user.id, False)

await message.answer("Вы итак не подписаны.")

else:

db.update\_subscription(message.from\_user.id, False)

await message.answer("Вы отписаны от рассылки.")

#погода

@dp.message\_handler(commands = ['weather'])

async def weather(message: types.Message):

await message.answer("Введите город:")

@dp.message\_handler(content\_types=["text"])

async def get\_city(message):

city = message.text # сохраняем данные

await message.answer(geo.weatherd(city))

@dp.message\_handler(commands=['start'])

async def buttons(message: types.Message):

await message.answer('',reply\_markup=kb.markup3)

# запускаем бота

if \_\_name\_\_ == '\_\_main\_\_':

executor.start\_polling(dp, skip\_updates=True)

Приложение 2. Модуль виртуальной клавиатуры.

#==============================

#Этот модуль виртуальной клавиатуры

#==============================

from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup, ReplyKeyboardRemove, KeyboardButton, ReplyKeyboardMarkup

button1 = InlineKeyboardButton(" ", callback\_data='btn1')

button2 = InlineKeyboardButton(" ", callback\_data='btn2')

button3 = InlineKeyboardButton(" ", callback\_data='btn3')

button4 = InlineKeyboardButton(" ", callback\_data='btn4')

button5 = InlineKeyboardButton(" ", callback\_data='btn5')

button6 = InlineKeyboardButton(" ", callback\_data='btn6')

button7 = InlineKeyboardButton(" ", callback\_data='btn7')

button8 = InlineKeyboardButton(" ", callback\_data='btn8')

button9 = InlineKeyboardButton(" ", callback\_data='btn9')

button10 = KeyboardButton('/weather')

button11 = KeyboardButton('/subscribe')

button12 = KeyboardButton('/unsubscribe')

markup3 = ReplyKeyboardMarkup().add(

button10, button11, button12

)

inline\_kb1 = InlineKeyboardMarkup()

inline\_kb1.add(

button1, button2, button3, button4, button5, button6, button7, button8, button9

)

# Приложение 3. Модуль погоды

#===================================================================

#С помощью этого модуля можно узнать текущую погоду в городе

#===================================================================

import pyowm

def weatherd(city):

owm = pyowm.OWM('a7a5f151b3845f1d0a5979f764dbb267')

mgr = owm.weather\_manager()

observation = mgr.weather\_at\_place(city + ',rus')

w = observation.weather

temp = w.temperature('celsius')["temp"]

wind = w.wind()["speed"]

hum = w.humidity

answer = "В городе " + city + " сейчас " + w.detailed\_status+ "\n---------------------"

answer += "\nТемпература: " + str(temp) + "°C" + "\n---------------------" + "\nСкорость ветра: " + str(wind) + "м/с." + "\n---------------------" + "\nВлажность: " + str(hum) + "%" + "\n---------------------"

return answer

# Приложение 4. Модуль БД

#===================================================================

#Модуль позволяет взаимодействовать с базой данных бота

#===================================================================

import sqlite3

class SQLighter:

def \_\_init\_\_(self, database):

"""Подключаемся к БД """

self.connection = sqlite3.connect(database)

self.cursor = self.connection.cursor()

def get\_subscriptions(self, status = True):

"""Получаем всех активных подписчиков бота"""

with self.connection:

return self.cursor.execute("SELECT \* FROM `subscriptions` WHERE `status` = ?", (status,)).fetchall()

def subscriber\_exists(self, user\_id):

"""Проверяем, есть ли уже пользователь в базе"""

with self.connection:

result = self.cursor.execute('SELECT \* FROM `subscriptions` WHERE `user\_id` = ?', (user\_id,)).fetchall()

return bool(len(result))

def add\_subscriber(self, user\_id, status = True):

"""Добавляем нового подписчика"""

with self.connection:

return self.cursor.execute("INSERT INTO `subscriptions` (`user\_id`, `status`) VALUES(?,?)", (user\_id,status))

def update\_subscription(self, user\_id, status):

"""Обновляем статус подписки пользователя"""

with self.connection:

return self.cursor.execute("UPDATE `subscriptions` SET `status` = ? WHERE `user\_id` = ?", (status, user\_id))

def close(self):

"""Закрываем соединение с БД"""

self.connection.close()