



Государственное бюджетное образовательное учреждение высшего образования  
Московской области

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ**

---

Колледж космического машиностроения и технологий

## КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка бота для Telegram: включаемые модули»

Выполнил студент

Звонарев Данила Александрович

Группа П1-17

\_\_\_\_\_ (Подпись)

\_\_\_\_\_ (Дата сдачи работы)

Проверил преподаватель

Гусятинер Леонид Борисович

\_\_\_\_\_ (Подпись)

\_\_\_\_\_ (Оценка)

Королёв 2020 г.

## Оглавление

Введение .....	3
1. Теоретическая часть .....	4
1.1. Описание предметной области.....	4
1.2. О Ботах.....	6
1.3. Описание существующих разработок.....	8
1.3.1.Бот “Шеф-повар” .....	9
1.3.2.Бот “Транспорт Минск” .....	10
2. Проектная часть.....	11
2.1. Диаграмма прецедентов.....	11
2.2. Выбор инструментов.....	12
2.3. Проектирование сценария.....	13
2.4. Диаграмма классов.....	14
2.5. Описание главного модуля .....	15
2.6. Описание спецификаций к модулям .....	17
2.7. Описание спецификаций к модулям .....	18
2.7.1.Keyboard.py.....	18
2.7.2.Geo.py.....	19
2.7.3.sqlighter.py .....	20
2.8. Описание тестовых наборов модулей .....	21
2.9. Описание применения средств отладки.....	24
2.10. Анализ оптимальности использования памяти и быстродействия .....	26
3. Эксплуатационная часть.....	27
3.1. Руководство оператора.....	27
Заключение.....	34
Список литературы и интернет-источников.....	35
Приложение 1. Код главного модуля bot.py.....	36
Приложение 2. Код модуля экранной клавиатуры keyboard.py.....	38
Приложение 3. Код модуля базы данных sqlighter.py. ....	39
Приложение 4. Код модуля работы с погодой geo.py.....	40

## **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.

## **1. Теоретическая часть**

### **1.1. 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 пишете 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.

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.

Telegram — кроссплатформенный мессенджер, позволяющий обмениваться сообщениями и медиафайлами многих форматов. Используются проприетарная серверная часть с закрытым кодом, работающая на мощностях нескольких международных компаний и несколько клиентов с открытым исходным кодом, в том числе под лицензией GNU GPL.

Количество ежемесячных активных пользователей сервиса, по состоянию на конец апреля 2020 года, составляет более 400 млн человек. Такими темпами к концу 2020 года количество пользователей мессенджером превысит 500 миллионов.

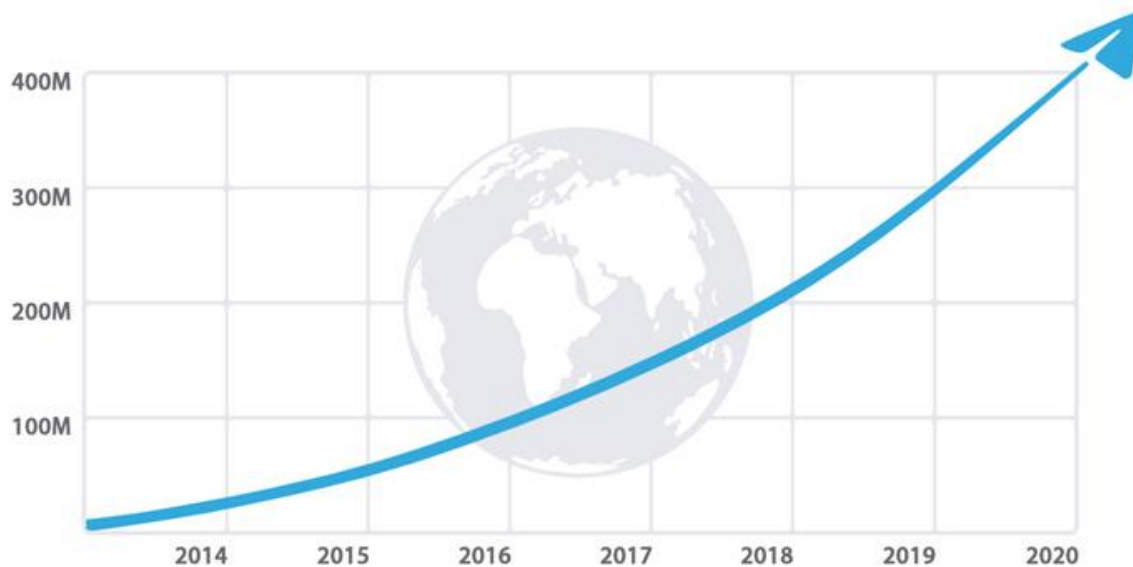


Рисунок 1. График роста аудитории Telegram

Средний возраст аудитории в России составляет 20 – 35 лет. Пользователи тратят от 10 до 20 минут ежедневно на этот мессенджер. Помимо стандартного обмена сообщениями в диалогах и группах, мессенджер выступает как файлообменник, с возможностью вести свой блог и создавать ботов.

## 1.2. О Ботах

**Бот** — специальная программа, выполняющая автоматически или по заданному расписанию какие-либо действия через интерфейсы, предназначенные для людей.

**Chat-bot (чат-бот)** – это виртуальный собеседник для мессенджера, который общается с пользователями посредством сообщений и с дополнительным функционалом.

**AI (Artificial intelligence)** - искусственный интеллект, используется для сокращения издержек и повышения качества услуг. Популярными направлениями являются распознавание голоса и текстовое общение посредством ботов.

Существуют боты для приема заявок на доставку еды, автоматических закупок на торговой бирже, рассылок рекламы. Такие боты позволяют увеличить прибыль компаний и сэкономить средства мелким предпринимателям, за счёт автоматизации и сокращения персонала.

Чат-боты в мессенджерах получили большое распространение особенно в Telegram ввиду наличия API в открытом доступе.

Для написания чат-бота был выбран **Aiogram API**. Он выступает в роли обёртки над **Telegram Bot API** ради облегчения дальнейшей разработки, за счёт использования готовых методов и объектов данной библиотеки. Что бы взаимодействовать с API необходимо получить ключ к нему.

**Ключ API** – это секретный код, который идентифицирует определенную учетную запись и позволяет использовать методы, для которых он необходим. В случае Bot API ключ выступает как путь, по которому можно обратиться, а в случае **PyOWM**, ключ API выступает как параметр для запроса погоды.

Для того, чтобы бот функционировал в Telegram, нужно его создать через специального мета-бота **BotFather (@BotFather)**. Нужно добавить его через поиск, в клиенте телеграмма. Список его команд можно получить, написав в чате с ним команду /help. Для создания нового бота нужно написать

команду `/newbot` и в следующем сообщении передать название бота (должно заканчиваться словом `bot`). В ответ придет сообщение с API ключом.

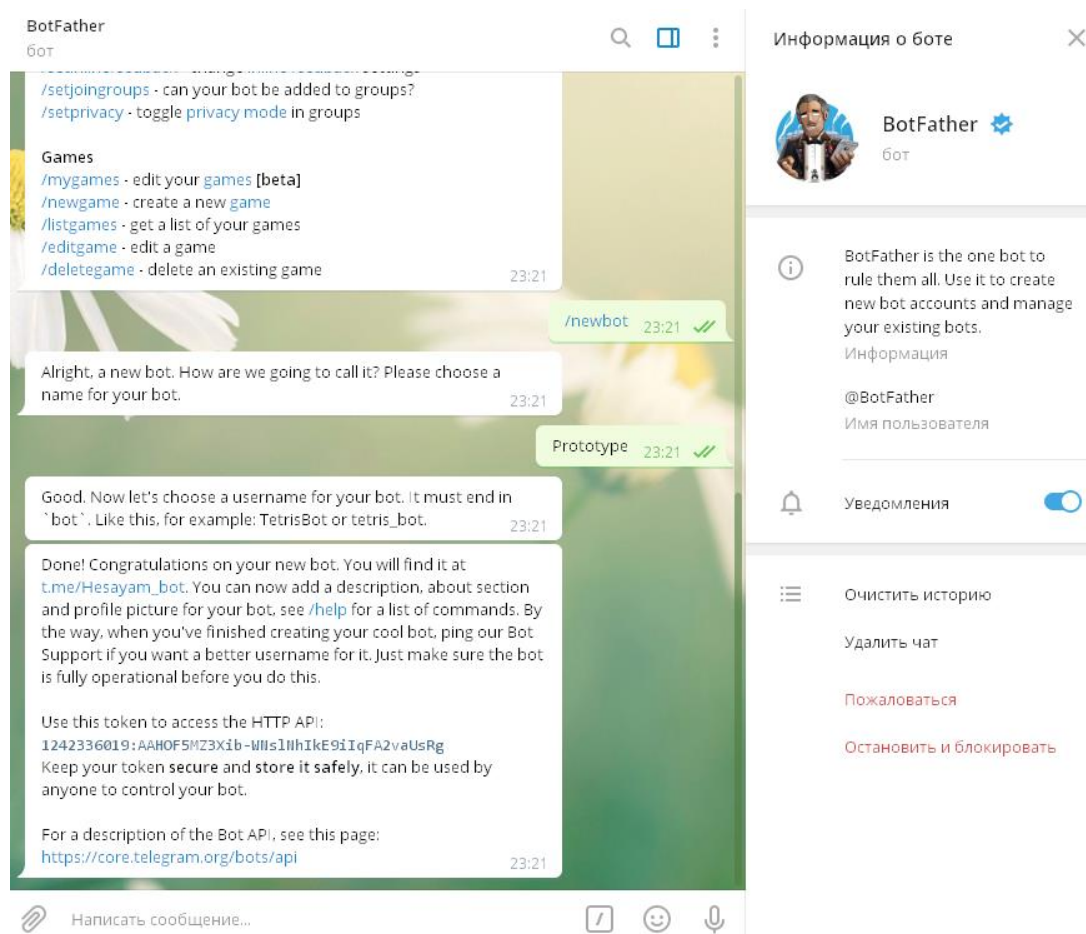


Рисунок 2. Создание бота в Telegram

Основной функцией бота было решено выбрать показ погоды в искомом городе. Для этого хорошо подходил **pyowm API** данная библиотека обладает широким функционалом, и постоянно обновляется.

Для работы с ней также необходимо получить ключ API, ключ выдается после регистрации на сайте данного сервиса в личном кабинете. С помощью данной библиотеки можно узнать прогноз погоды на дни недели в населенном пункте, погоду по географическим координатам. Она является open source проектом и предоставляется по MIT лицензии.

Что бы хранить список подписчиков чат-бота использовалась **СУБД SQLite**. Этой компактной базы данных оказалось вполне достаточно. Для работы с БД необходима библиотека `sqlite3`, она позволяет установить связь с БД, создавать, редактировать и удалять записи оттуда.

### **1.3. Описание существующих разработок**

В этом разделе рассмотрены некоторые уже имеющиеся программы для управления отелями, а также цели таких программ и требования.

Интерфейс таких виртуальных собеседников довольно разнообразен, но чаще всего выполнен в упрощенном формате.

#### **Цели использования чат-ботов:**

- Автоматизация работы текстовых каналов.
- Рассылка новостей и другой информации.
- Рассылка рекламы.

#### **Недостатки ботов:**

- Необходимо программное обеспечение
- Нужен человек, обслуживающий бота
- Боты не самостоятельны и обладают ограниченным функционалом.



### 1.3.1. Бот “Шеф-повар”

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

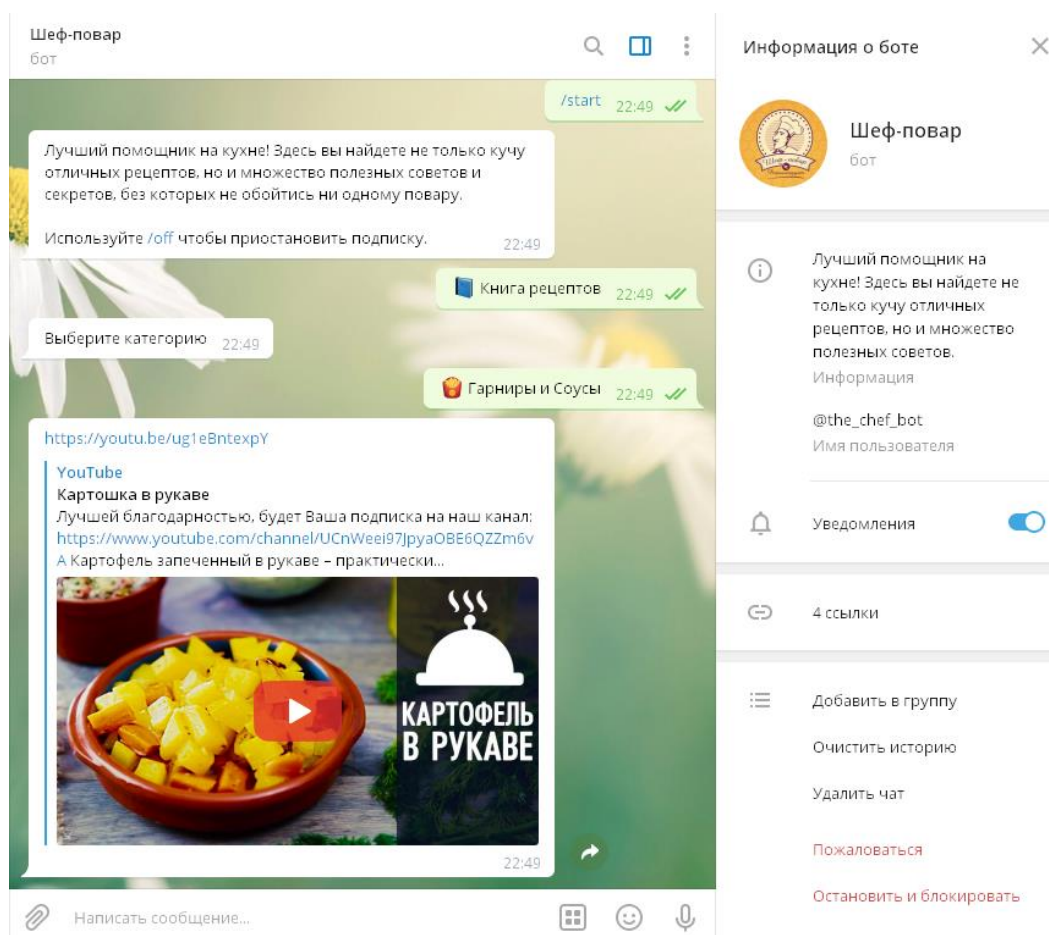


Рисунок 3. Бот Шеф-повар

Ответом является ссылка на видеоролик с рецептом данного блюда, что очень удобно.

### 1.3.2. Бот “Транспорт Минск”

Этот чат-бот предназначен для просмотра расписания автобусов

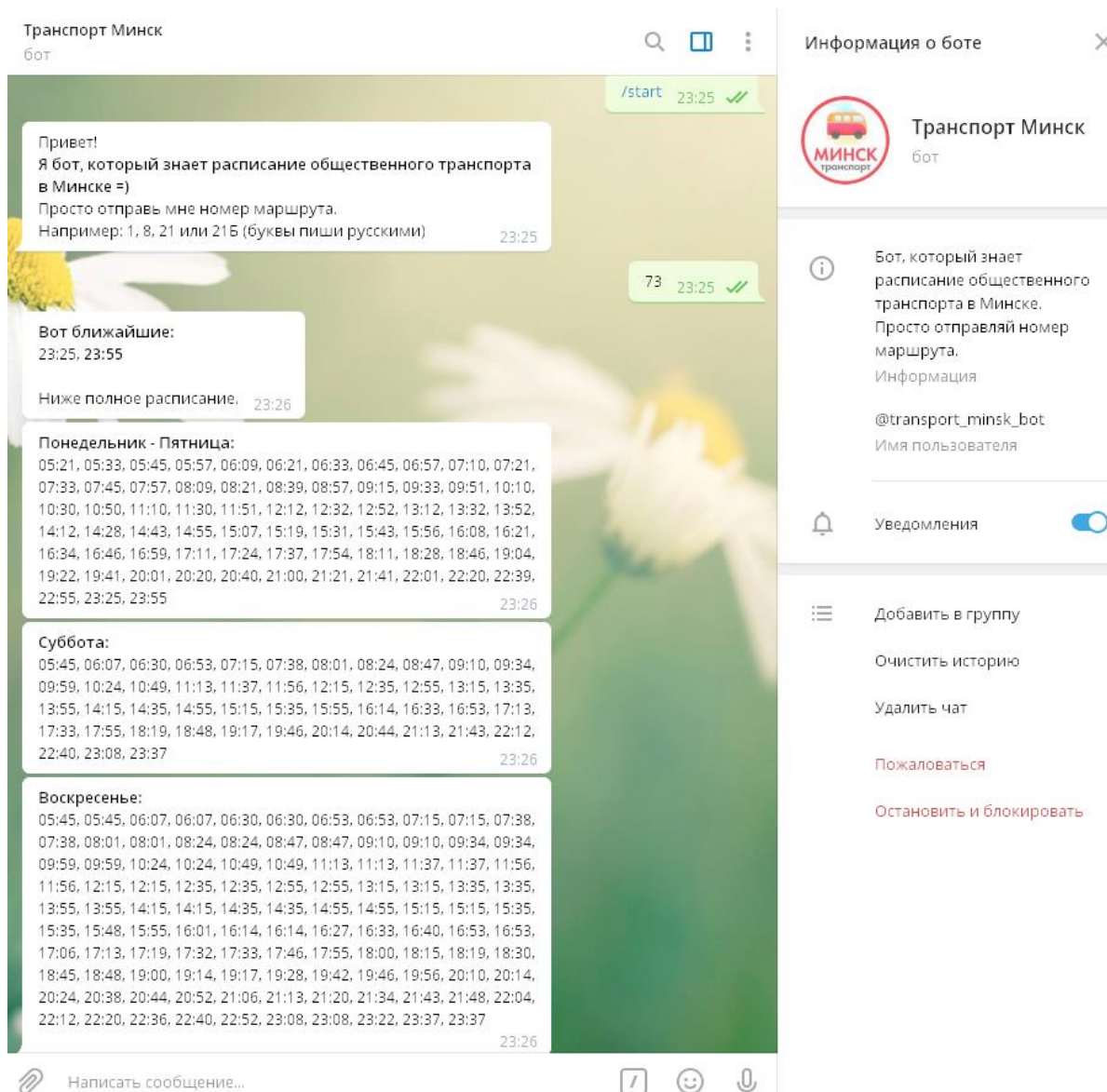


Рисунок 4. Бот для расписания автотранспорта

Данный бот способен выдавать расписание по определенным дням недели, для конкретного маршрута движения, а также показывают ближайшие автобусы к определенной остановке. Такие боты удобны и многофункциональны.

## 2. Проектная часть

### 2.1. Диаграмма прецедентов

В этом разделе представлена диаграмма прецедентов. На диаграмме (Рисунок 5) показаны возможные действия пользователя.

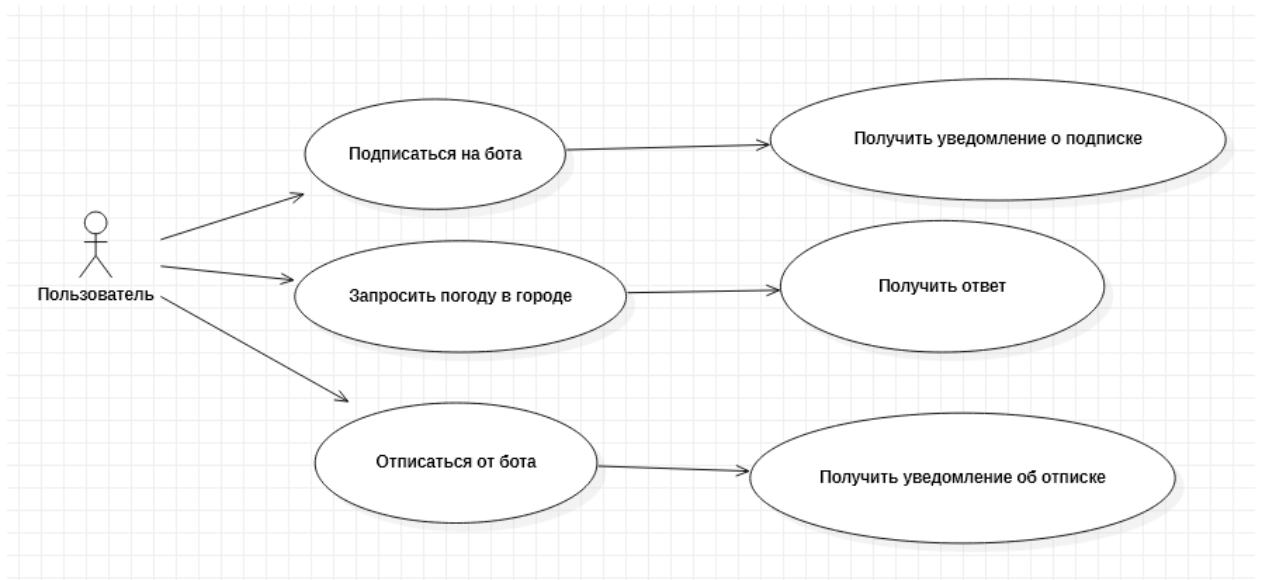


Рисунок 5. Диаграмма "Пользователь"

Взаимодействие с чат-ботом ведется посредством виртуальной клавиатуры. У нее есть три действия:

- Подписаться на бота
- Запросить погоду в городе
- Отписаться от бота

## 2.2. Выбор инструментов

При выборе инструментов было проведено сравнение по критериям, представленных в таблице 1.

Степень важности критерия выбиралась из: низкая, ниже средней, средняя, ниже высокой, высокая.

Таблица 1. Критерии выбора инструмента.

Критерий	Участие в корпоративном проекте	Простота сопровождения	Наличие библиотек	Наличие документации на русском языке	Скорость разработки
Важность критерия	Высокая	Средняя	Высокая	Ниже средней	Ниже высокой

Исходя из этих критериев, я сравнил 3 языка программирования от 0 до 10 баллов за критерий.

Таблица 2. Оценка языков программирования.

Критерий/Язык программирования	Haskell	Python	Java
Участие в корпоративном проекте	5	8	4
Простота сопровождения	7	10	5
Наличие библиотек	7	10	8
Наличие документации на русском языке	4	6	5
Скорость разработки	8	10	5
Итого баллов	31	44	27

По результатам сравнения был выбран язык программирования Python.

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>

### 2.3. Проектирование сценария

В данном разделе приведен сценарий использования программы пользователем.

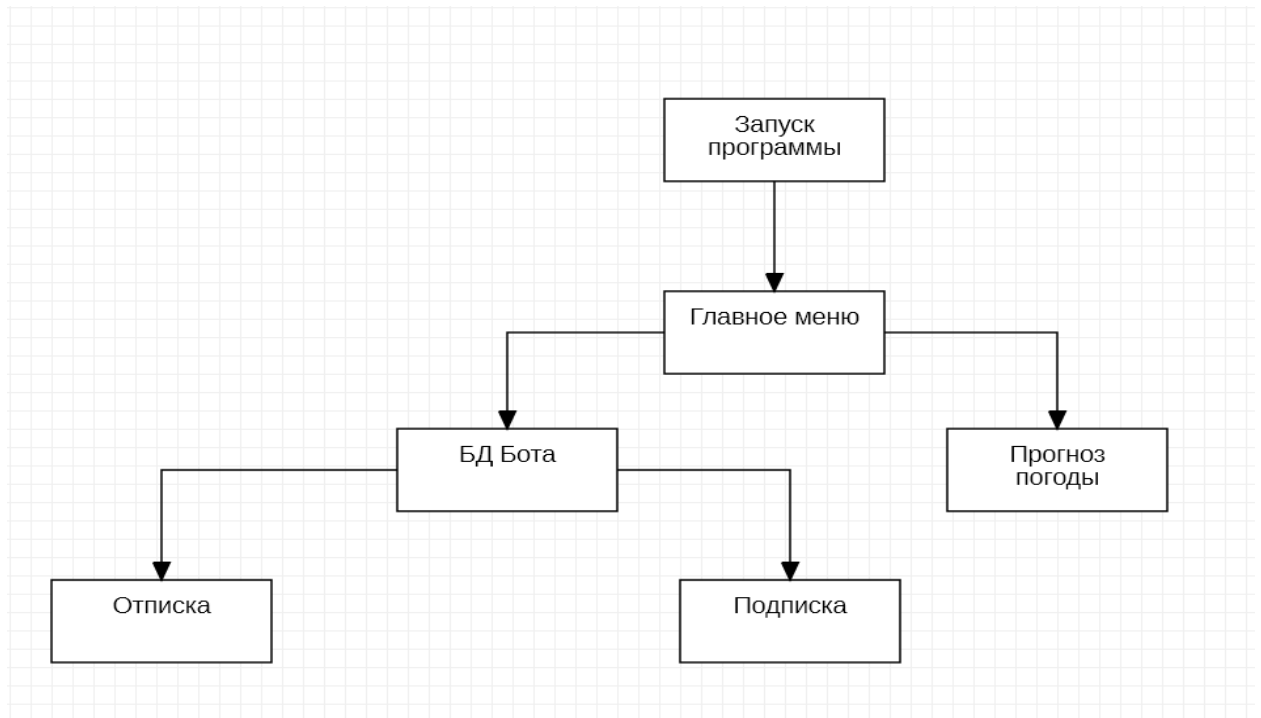


Рисунок 6. Сценарий использования

Пользователь после запуска программы может выполнить 3 действия: подписаться на чат-бота, отписаться от него и запросить погоду в своем регионе. Выход из чата вызывает сценарий завершения работы.

## 2.4. Диаграмма классов

В данном разделе представлены все классы, использующиеся в проекте, а также их отношения между собой.

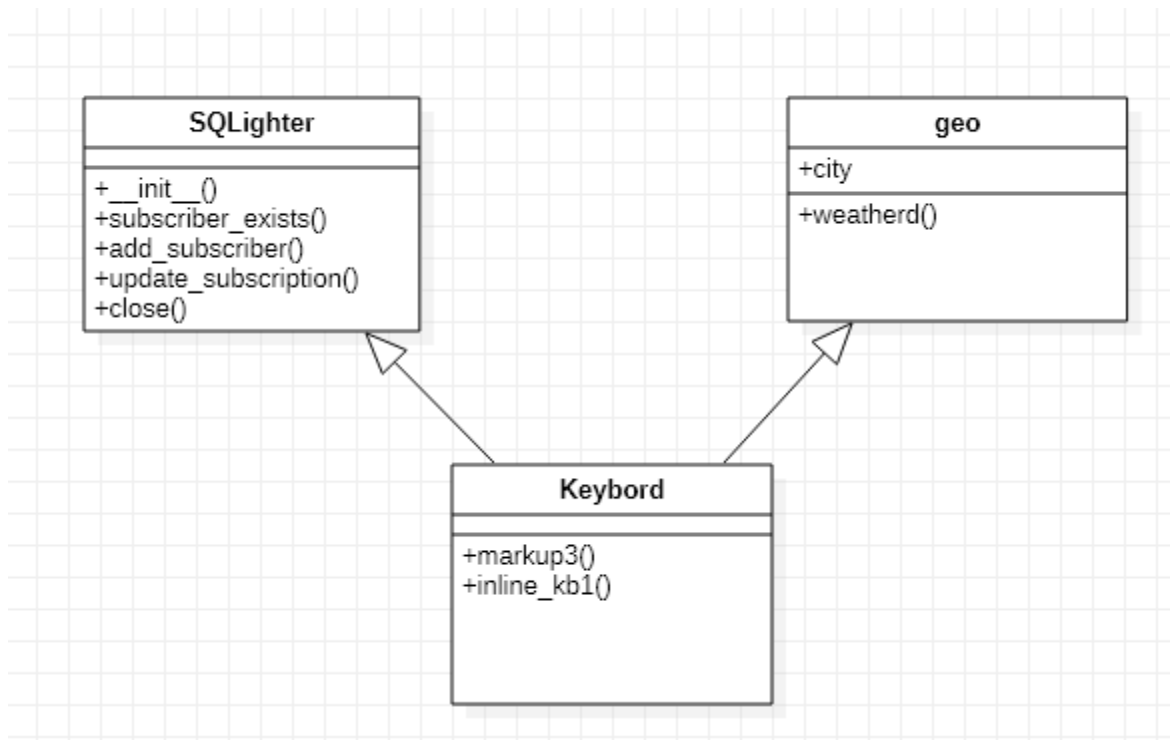


Рисунок 7. Диаграмма классов

Класс SQLiter содержит публичные методы для работы с базой данных подписчиков:

get\_subscription(), который позволяет получить список активных подписчиков чат-бота

subscriber\_exists(), проверяет наличие пользователя в базе данных

add\_subscriber(), записывает нового подписчика в базу данных

update\_subscribtion(), обновляет статус подписки пользователя

close().

Класс geo содержит метод weatherd(), который позволяет запросить погоду в городе.

Класс Keyboard содержит методы markup3(), inline\_kb1(), которые необходимы для вывода экранной виртуальной клавиатуры для управления программой через главный модуль

## 2.5. Описание главного модуля

Главный модуль отвечает за функционал программы и подключает 3 второстепенных модуля программы.

В главный модуль также импортируется еще 4 библиотеки, одна из которых описывает работу бота через Aiogram API).

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

В Листинге 1 представлена Обработка событий в программе:

### Листинг 1. Обработка всех событий в программе.

```
#Булатников Илья П1-17
#команда активации подписки
@dp.message_handler(commands=['subscribe'])
#если сообщение от пользователя 'subscribe' делать дальнейшие команды

#async def — Сопрограмма (coroutine) асинхронная функция
async def subscribe(message: types.Message):

    #проверка на наличие подписки у пользователя
    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)

async def subscribe(message: types.Message):

    #проверка на наличие подписки у пользователя
    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)
        #Ключевое слово await приостанавливает выполнение текущей
        #сопрограммы (coroutine) и вызывает message.answer.
        await message.answer("Вы подписались на бота")

#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):
        # сохраняем данные о городе
        city = message.text
        # печать в чат погоды
        await message.answer(geo.weatherd(city))

#вывод кнопки команд на экран
@dp.message_handler(commands=['start'])
async def buttons(message: types.Message):
```

```

        await message.answer('btn', reply_markup=kb.markup3)

#скрытие с экрана кнопок
@dp.message_handler(commands=['remove'])
async def removekb(message: types.Message):
    await message.answer('remove_kb', reply_markup=kb.ReplyKeyboardRemove())

#запуск бота
if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)

```

Данный фрагмент кода отслеживает все действия пользователя и в зависимости от конкретного действия вызывает нужный метод.

В Листинге 2 представлена Обработка событий в программе:

### **Листинг 2. Обработка команды start в программе.**

```

#вывод кнопки команд на экран
@dp.message_handler(commands=['start'])
async def buttons(message: types.Message):
    await message.answer('btn', reply_markup=kb.markup3)

```

Данный метод выводит на экран пользователя виртуальную клавиатуру.



## 2.6. Описание спецификаций к модулям

В данном разделе описаны спецификации к модулям. Спецификация к модулям заключается в использовании методов библиотек и API aiogram, pyowm, sqlite3.

Методы:

- `await message.answer` – отправка сообщения в чат пользователю от бота.
- `start_polling` – запускает бота на компьютере.
- `weather_manager()` – объект класса погода.
- `wind` – возвращает параметр скорость ветра.
- `weather_at_place()` – определение погоды в введенном городе.
- `humidity` – возвращает параметр влажность воздуха.
- `temperature` – возвращает параметр температура воздуха.
- `detailed_status` – возвращает словесное описание погоды.

## 2.7. Описание модулей

Кроме главного модуля программа содержит три модуля. Они расширяют функционал основного модуля.

### 2.7.1. Keybord.py

Модуль клавиатуры нужен для вывода на экран меню из кнопок для взаимодействия с программой.

#### Листинг 3. Модуль клавиатуры и меню.

```
# Звонарев Данила П1-17
#импорт модулей для работы с клавиатурой и кнопками
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup,
ReplyKeyboardRemove, KeyboardButton, ReplyKeyboardMarkup

#кнопки для меню
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')

inline_kb1 = InlineKeyboardMarkup()
#печать в чат поля 3*3 из кнопок для будущего меню
inline_kb1.add(
    button1, button2, button3, button4, button5, button6, button7, button8,
    button9
)
#кнопки для виртуальной клавиатуры
button10 = KeyboardButton('/weather')
button11 = KeyboardButton('/subscribe')
button12 = KeyboardButton('/unsubscribe')
#вывод на экран виртуальной клавиатуры с кнопками
markup3 = ReplyKeyboardMarkup().row(
    button10, button11, button12
)
```

## 2.7.2. Geo.py

Модуль погоды нужен для просмотра прогноза погоды в по введенному городу.

### Листинг 4. Модуль погоды.

```
# Звонарев Данила П1-17
import pyowm
#импорт модуля для работы с погодой

#погода
def weatherd(city):

    # инициализируем бота для работы с погодой
    owm = pyowm.OWM('a7a5f151b3845f1d0a5979f764dbb267')
    mgr = owm.weather_manager()
    # получение информации о погоде в городе
    observation = mgr.weather_at_place(city + ',rus')
    # погода
    w = observation.weather
    # температура
    temp = w.temperature('celsius')['temp']
    # скорость ветра
    wind = w.wind()['speed']
    # погода влажность
    hum = w.humidity
    # формирование ответа, выводимого в чат пользователю
    answer = "В городе " + city + " сейчас " + w.detailed_status + "\n-----
-----"
    answer += "\nТемпература: " + str(temp) + "°C" + "\n-----"
    " + "\nСкорость ветра: " + str(wind) + "м/с."
    answer += "\n-----" + "\nВлажность: " + str(hum) + "%" +
"\n-----"
    return answer
```

### 2.7.3. sqlighter.py

Данный модуль отвечает за работу с базой данных sqlight

Полный код модуля приведен в Приложение 3. В этом разделе приведены все методы, которые содержатся в класс SQLighter из модуля sqlighter .py.

#### Листинг 5. Методы класса SQLighter.

```
# Булатников Илья П1-17
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()
```

## 2.8. Описание тестовых наборов модулей

В этом разделе будут продемонстрированы результаты тестирования “черного ящика”.

Тест 1. Экранная клавиатура. Проверка кнопки Start

Действия: Нажать на кнопку Start.

Ожидаемый результат: появление меню из трёх кнопок.

Результат теста:

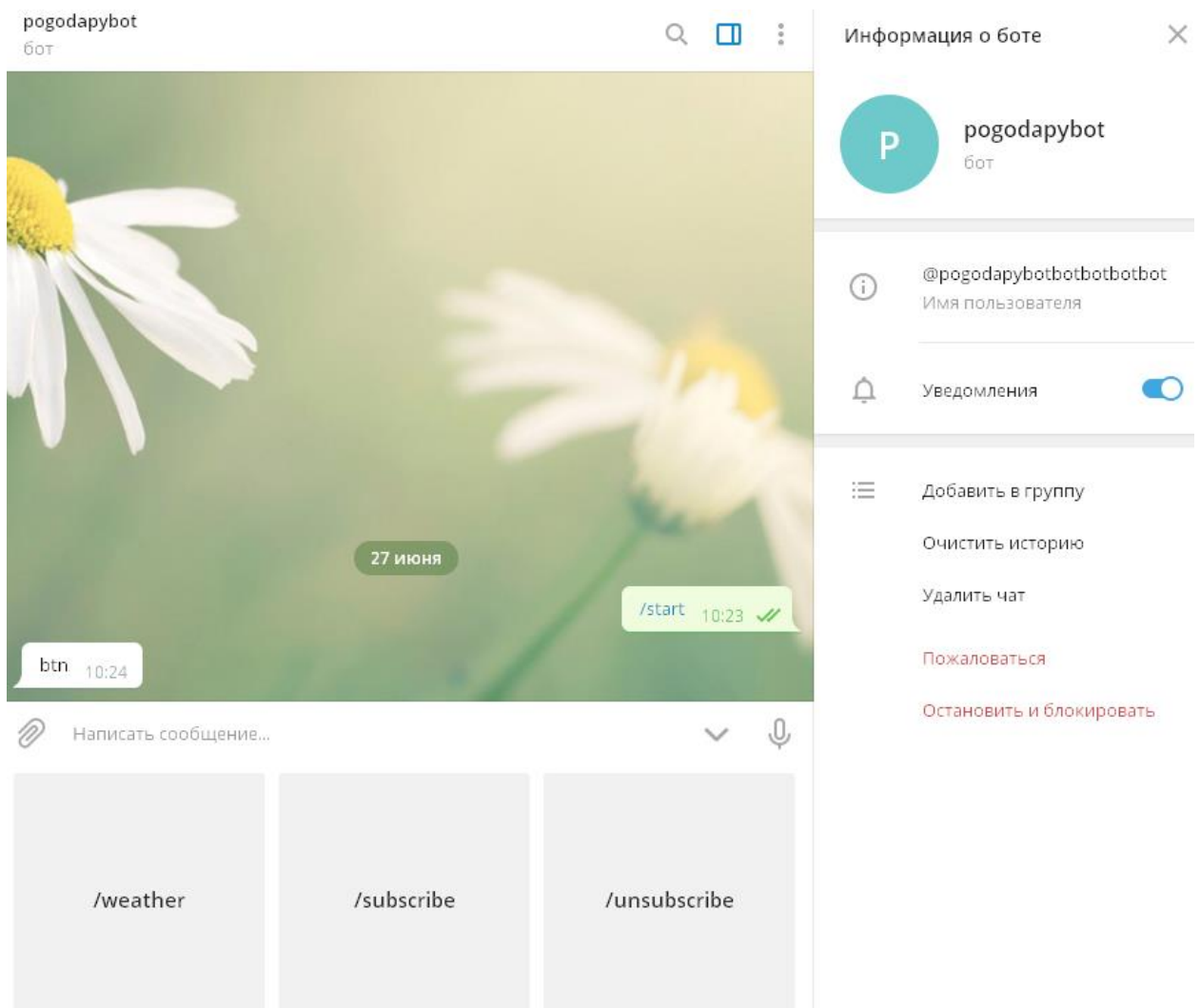


Рисунок 9. Экранная клавиатура

## Тест 2 Проверка погоды

Действия: Нажать на кнопку “/weather”, в ответ на появившееся предложение ввести город ввести необходимую информацию и нажать на кнопку Send.

Ожидаемы результат: Вывод на экран текущей погоды в выбранном городе.

Результат теста:

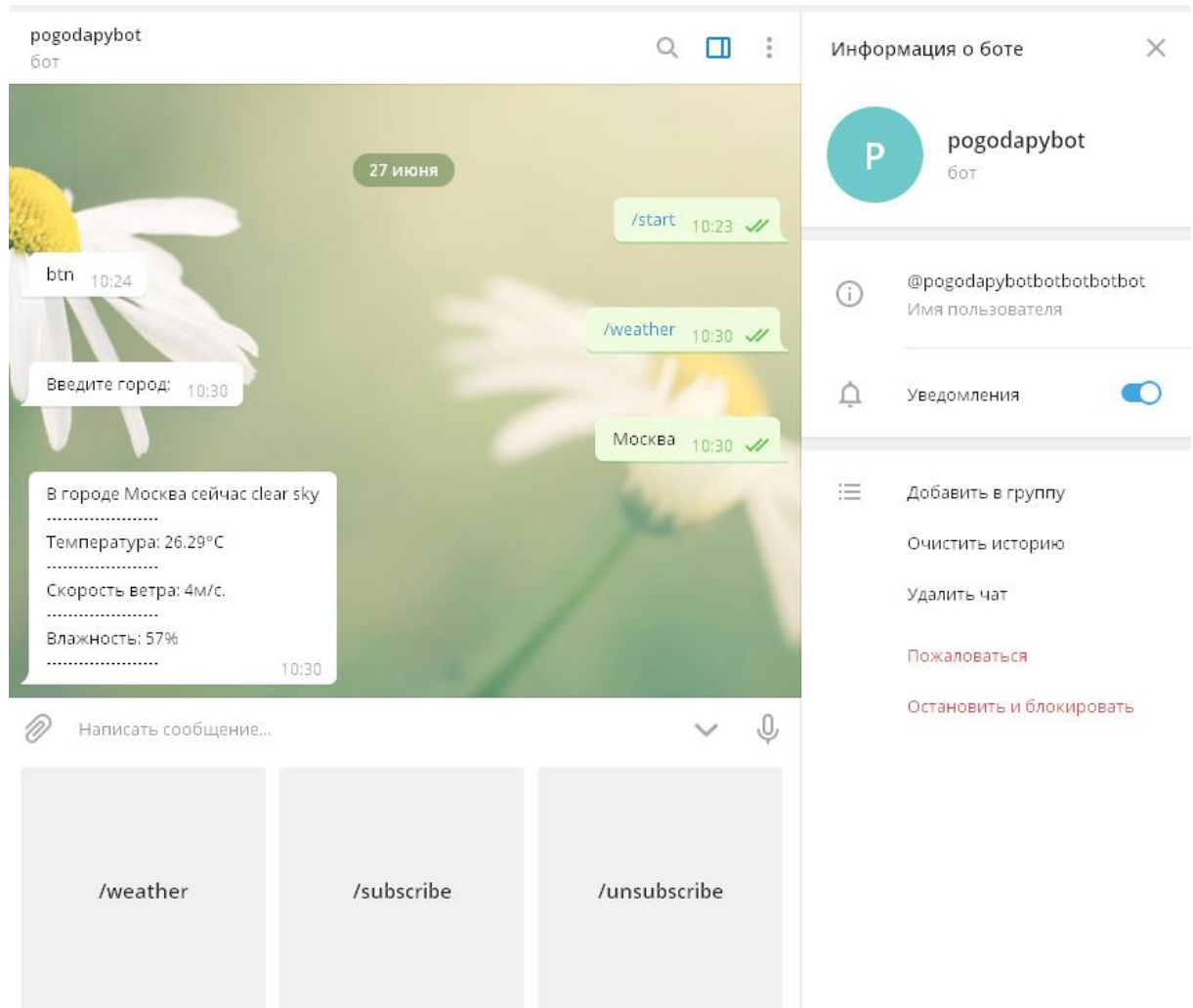


Рисунок 10. Погода в Москве

### Тест 3. Проверка подписки.

Действия: Нажать на кнопку “/subscribe”.

Ожидаемый результат: Сообщение о подписке.

Результат теста:

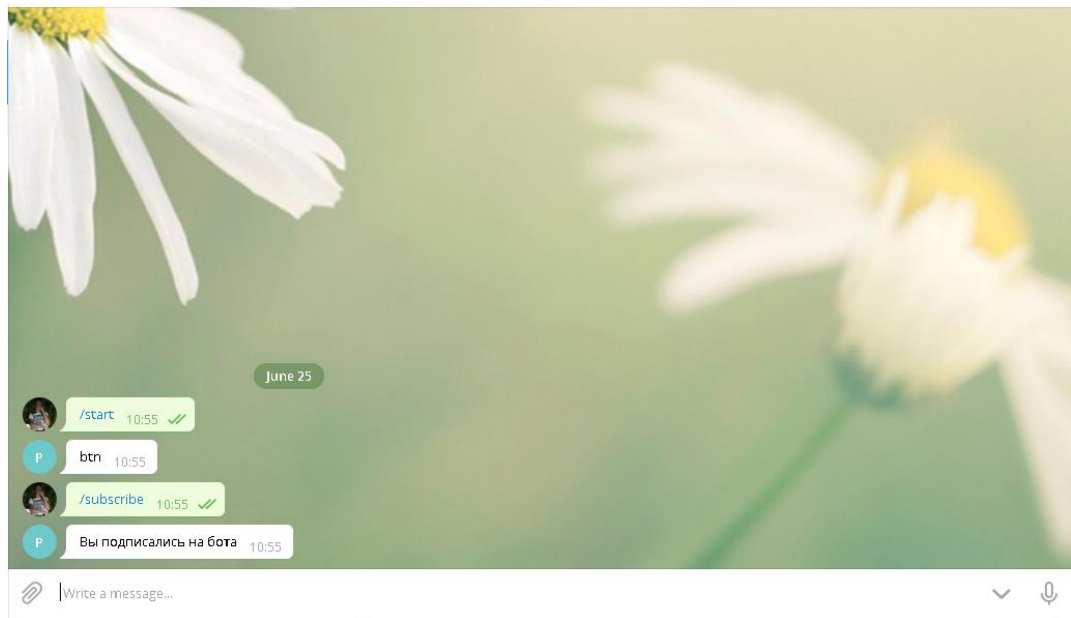


Рисунок 21. Уведомление о подписке

### Тест 4. Проверка отписки.

Действия: Нажать на кнопку “/unsubscribe”.

Ожидаемый результат: Сообщение об отписке.

Результат теста:

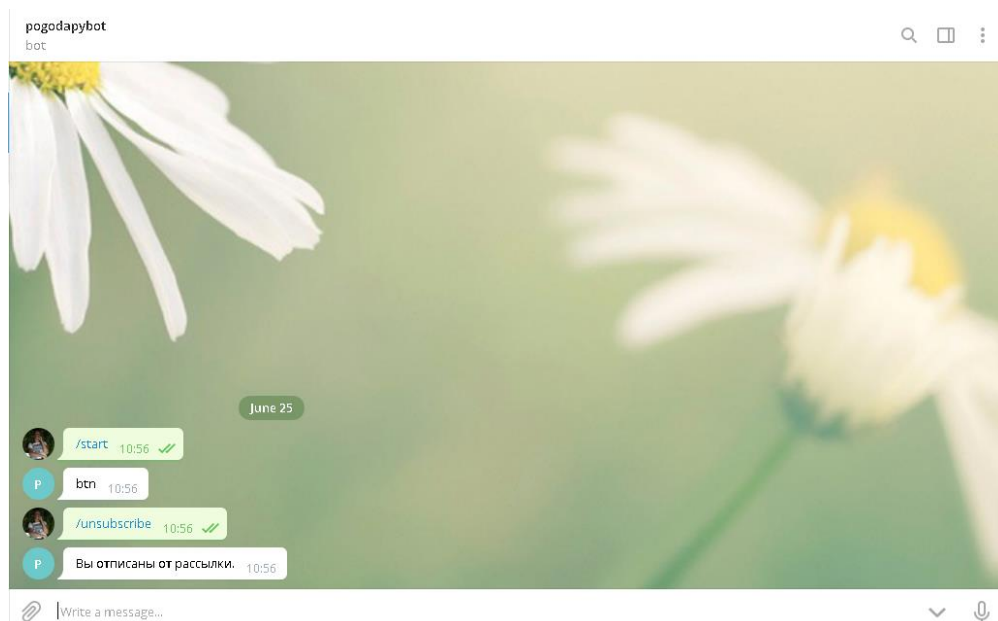


Рисунок 12. Уведомление о подписке

## 2.9. Описание применения средств отладки

В этом разделе показано умение применять средства отладки.

В ходе написания курсового проекта при попытке запустить программу было получено данное сообщение:

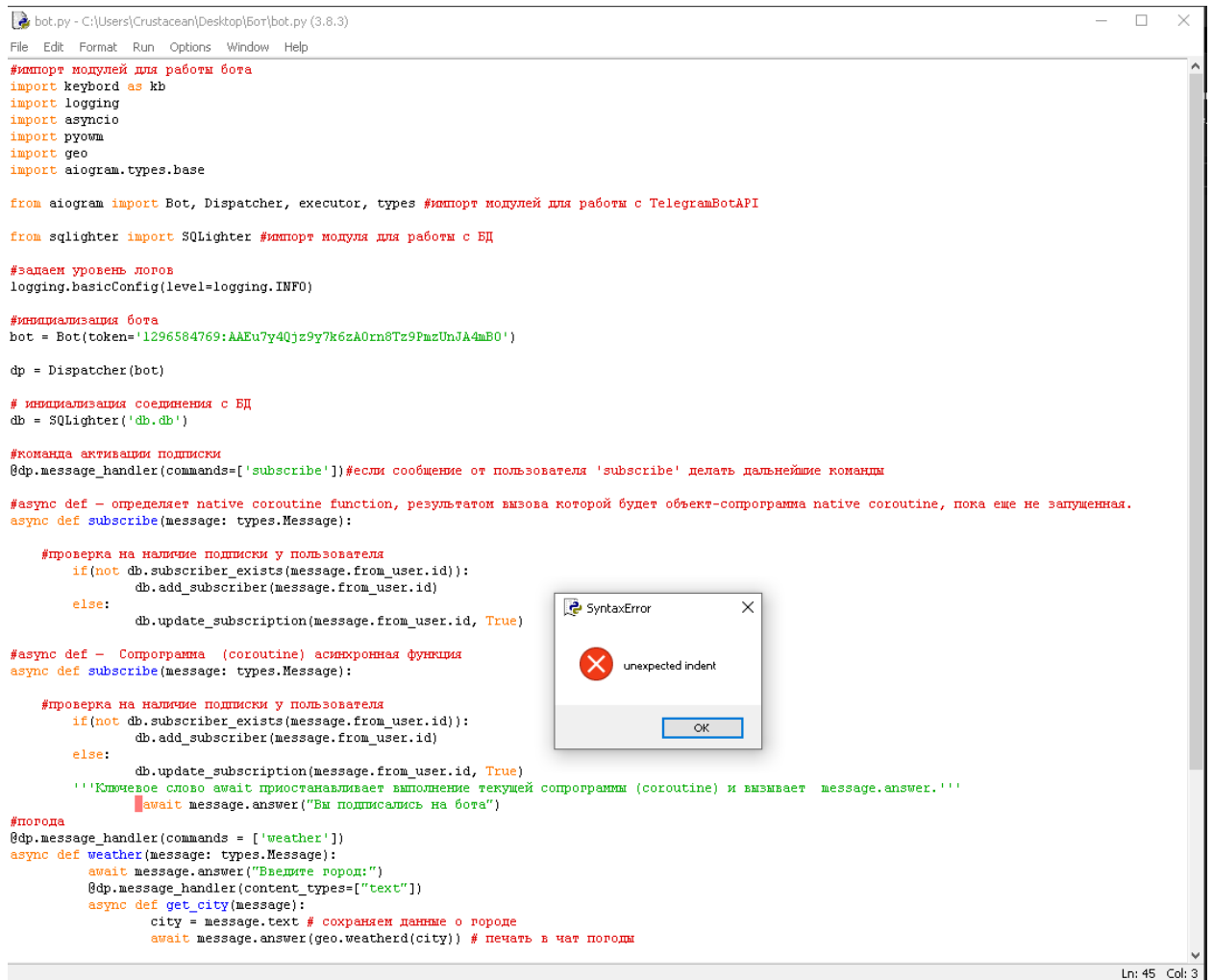


Рисунок 33. До применения средств отладки

После получения данного сообщения были просмотрены 43 и 44 строки модуля bot.py и была обнаружена ошибка, которая впоследствии была устранена, а после попытки запуска программы получено данное сообщение:



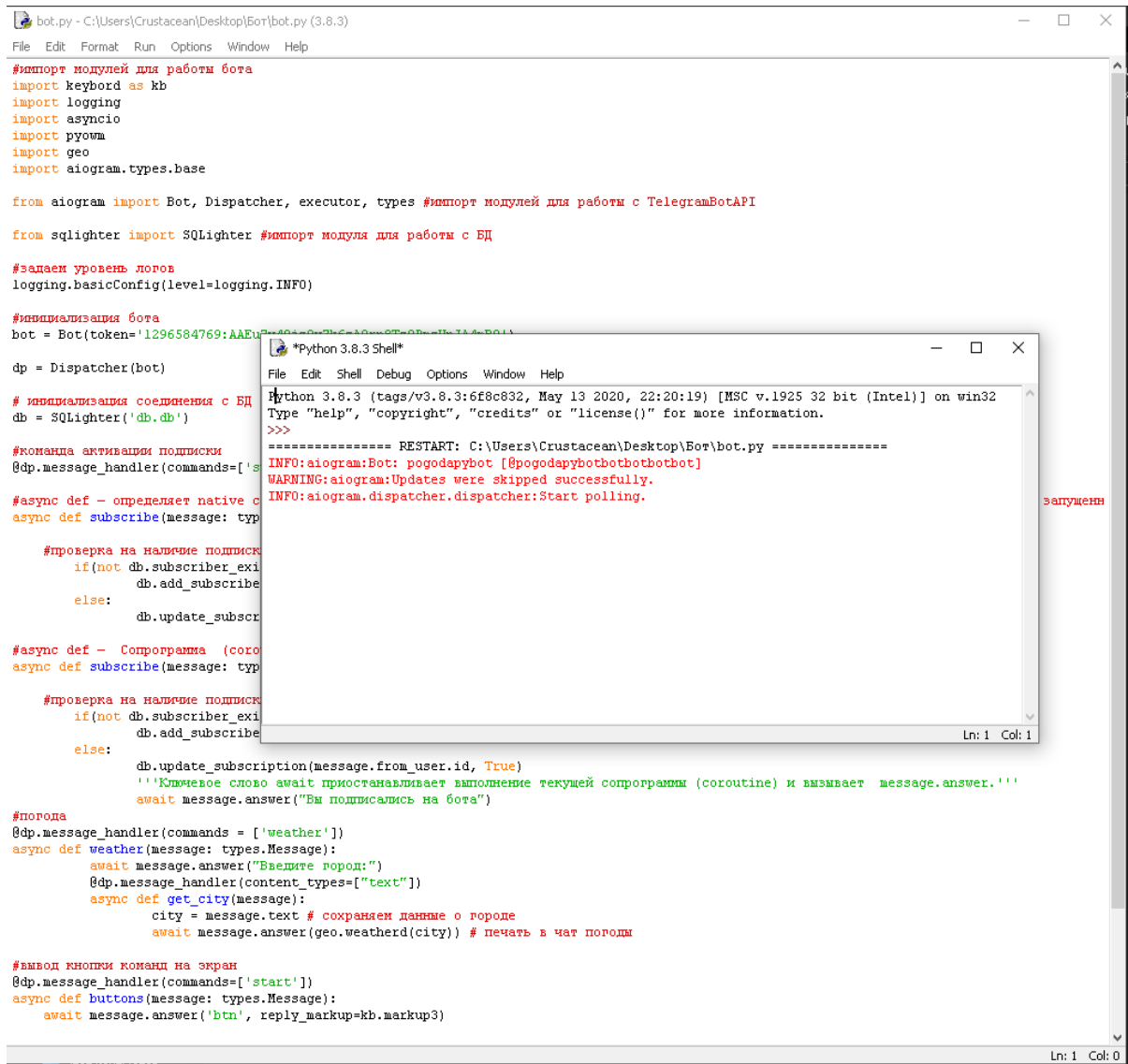


Рисунок 44. После применения средств отладки

Это означает что ошибка была устранена и скрипт запустился.

## 2.10. Анализ оптимальности использования памяти и быстродействия

В данном разделе будет проведен анализ оптимальности использования памяти и быстродействия программы.

Список принятых оптимальных решений:

1. Подключение некоторых модулей внутри функций/методов.

В данном проекте некоторые модули были подключены не в весь модуль, а только в функции/методы, которые его используют. Сделано это потому что работа с локальными объектами быстрее работы с глобальными объектами, к тому же импортироваться эти модули будут только при срабатывании этих функций что ускорит запуск программы. Пример такого импортирования виден Листинг 3 и Листинг 4.

2. Импортирование конкретных пакетов из модуля библиотеки.

В данном проекте некоторые модули были подключены не полностью, а только конкретные пакеты и объекты, которые используются внутри программы. Это сделано для экономии ресурсов вычислительной машины конкретизация импортируемых пакетов ускоряет запуск программы. Пример такого импортирования виден в Листинге 9 и Листинге 10

### Листинг 9. Импорт объектов пример 1

```
#импорт модулей для работы с клавиатурой и кнопками
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup,
ReplyKeyboardRemove, KeyboardButton, ReplyKeyboardMarkup
```

### Листинг 10. Импорт объектов пример 2

```
from aiogram import Bot, Dispatcher, executor, types
#импорт модулей для работы с TelegramBotAPI
```

### **3. Эксплуатационная часть**

#### **3.1. Руководство оператора**

##### **АННОТАЦИЯ**

В данном программном документе приведено руководство оператора по применению и эксплуатации программы «Pogoda bot», предназначенной для просмотра прогноза погоды в городах мира.

В данном программном документе, в разделе «Назначение программы» указаны сведения о назначении программы и информация, достаточная для понимания функций программы и ее эксплуатации.

В разделе «Условия выполнения программы» указаны условия, необходимые для выполнения программы (минимальный состав аппаратных и программных средств и т.п.).

В данном программном документе, в разделе «Выполнение программы» указана последовательность действий оператора, обеспечивающих загрузку, запуск, выполнение и завершение программы, приведено описание функций, формата и возможных вариантов команд, с помощью которых оператор осуществляет загрузку и управляет выполнением программы, а также ответы программы на эти команды.

Оформление программного документа «Руководство оператора» произведено по требованиям ЕСПД (ГОСТ 19.101-77 <sup>1)</sup>, ГОСТ 19.103-77 <sup>2)</sup>, ГОСТ 19.104-78\* <sup>3)</sup>, ГОСТ 19.105-78\* <sup>4)</sup>, ГОСТ 19.106-78\* <sup>5)</sup>, ГОСТ 19.505-79\* <sup>6)</sup>, ГОСТ 19.604-78\* <sup>7)</sup>).

---

<sup>1)</sup> ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов

<sup>2)</sup> ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов

<sup>3)</sup> ГОСТ 19.104-78\* ЕСПД. Основные надписи

<sup>4)</sup> ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам

<sup>5)</sup> ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом

<sup>6)</sup> ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению

<sup>7)</sup> ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом

## **1. Назначение программы**

### **1.1. Функциональное назначение программы**

Специальное программное обеспечение «Pogoda bot» используется для просмотра прогноза погоды.

### **1.2. Эксплуатационное назначение программы**

Специальное программное обеспечение «Pogoda bot» может эксплуатироваться на хостинге серверов или локальном компьютере где может использоваться как самостоятельная программа, или в купе с другим программным обеспечением.

### **1.3. Состав функций**

#### **1.3.1. Функция открытия меню.**

Эта функция позволяет вызвать на экран виртуальную клавиатуру с доступными действиями для пользователя

#### **1.3.2. Функция подписки на чат-бота**

Эта функция позволяет пользователя подписать на чат-бота, в результате чего информация о пользователе заносится в базу данных

#### **1.3.3. Функция отписки от чат-бота.**

Эта функция позволяет пользователю отписаться от чат-бота, в результате чего информация о пользователе стирается из базы данных

#### **1.3.4. Функция запроса погоды**

Эта функция позволяет узнать текущую погоду в введенном городе.

## **2. Условия выполнения программы**

### **2.1. Минимальный состав аппаратных средств**

ОС: Windows XP (32 бит) и новее

Процессор: Как минимум 1 ГГц или SoC.

ОЗУ: 128 МБ или больше.

Место на жестком диске: 300 МБ.

Видеоадаптер: DirectX версии не ниже 9 с драйвером WDDM 1.0.

Дисплей: 380 x500.

### **2.2. Минимальный состав программных средств**

Telegram messenger

SQLite3

### **2.3. Требование к персоналу (пользователю)**

Конечный пользователь программы должен обладать практическими навыками работы с графическим пользовательским интерфейсом операционной системы и мессенджером Telegram.

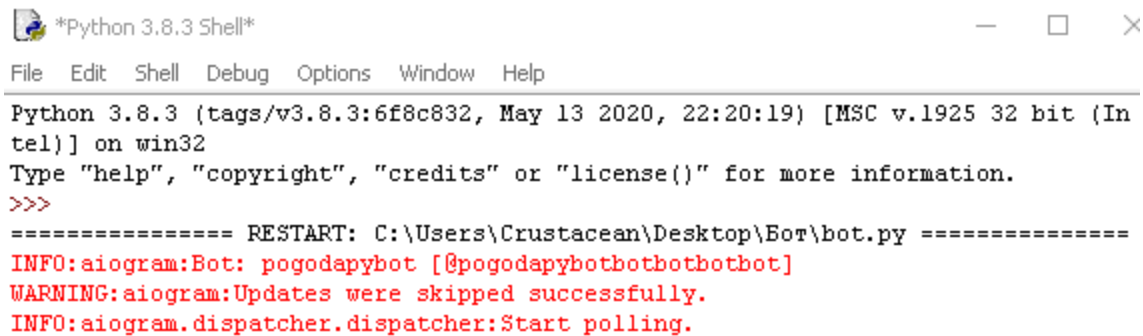
### 3. Выполнение программы

#### 3.1. Загрузка и запуск программы

Перед запуском программы необходимо установить:

- СУБД SQLite
- Telegram

Запустив программу «Pogoda bot», вы увидите в консоли сообщение:



```

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Crustacean\Desktop\Bot\bot.py =====
INFO:aiogram:Bot: pogodapybot [@pogodapybotbotbotbotbot]
WARNING:aiogram:Updates were skipped successfully.
INFO:aiogram.dispatcher.dispatcher:Start polling.
  
```

Рисунок 15. Уведомление о запуске бота

Это означает, что бот был запущен, и вы можете приступить к работе уже с самим чат-ботом в приложении Telegram.

#### 3.2. Выполнение программы

##### 3.2.1. Поиск чат-бота в списке каналов.

Запустите Telegram и в поиске каналов введите “@pogodapybotbotbotbotbot”

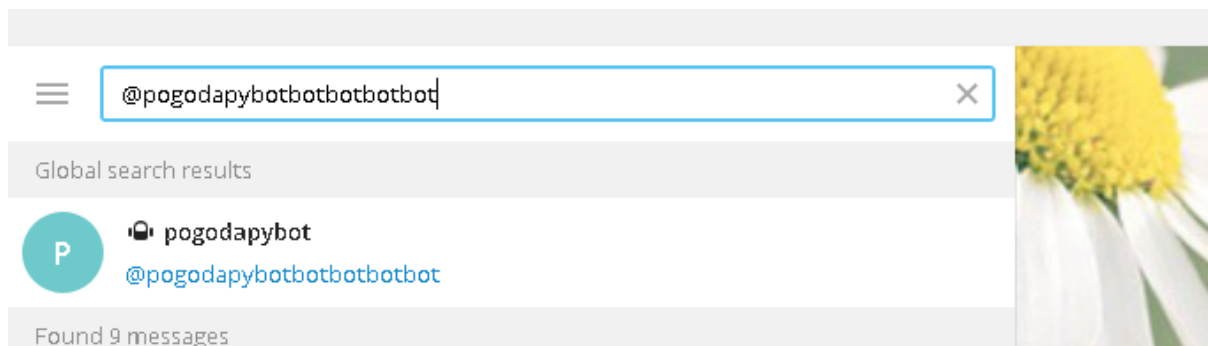


Рисунок 16. Поиск бота в списке каналов

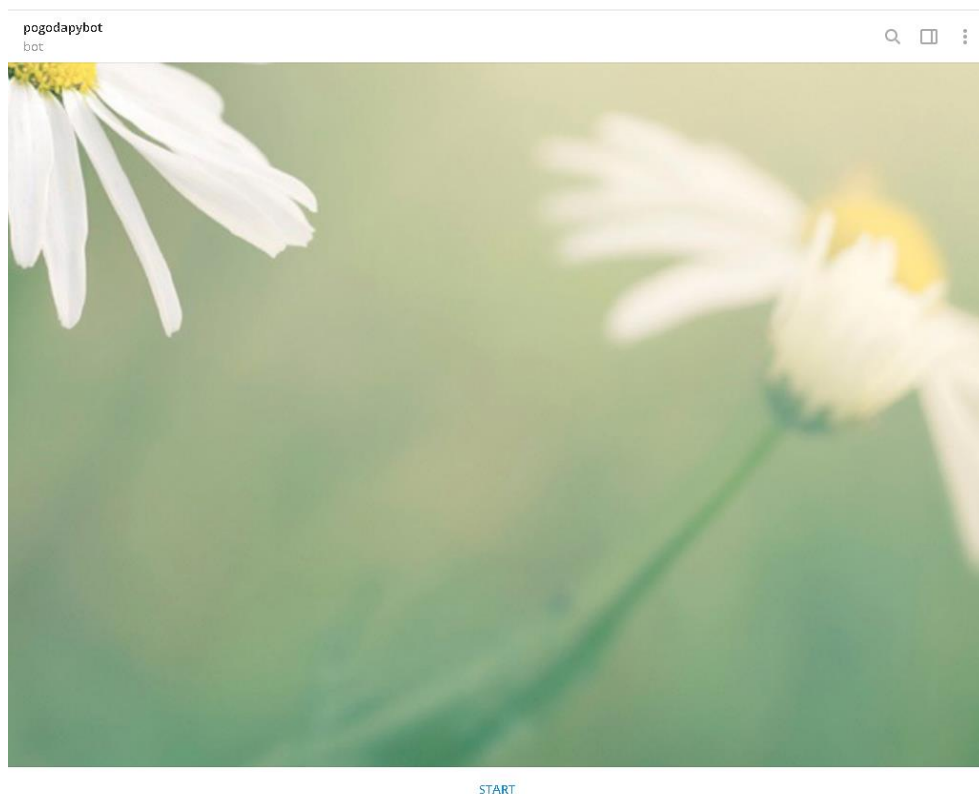


Рисунок 17. Начальное окно диалога чат-бота

### 3.2.2. Выполнение функции открытия меню.

Нажмите на кнопку Start находящуюся в окне диалога чат-бота.

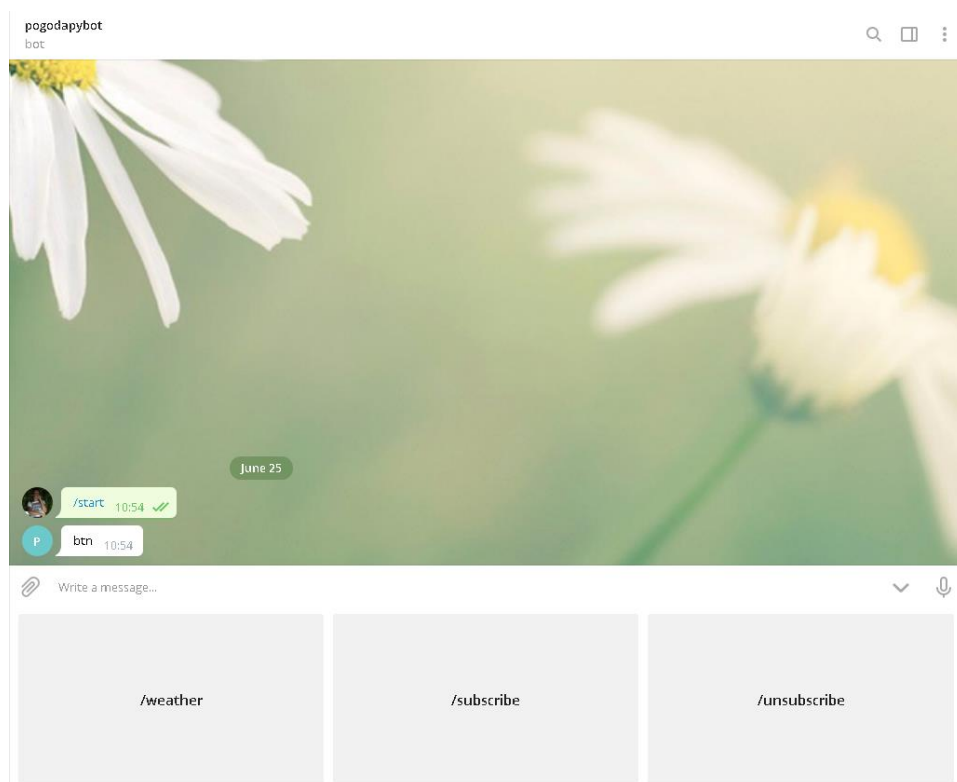


Рисунок 18. Экранная клавиатура с доступными командами

### 3.2.3. Выполнение функции подписки на чат-бота.

Нажмите на кнопку /subscribe находящуюся в окне диалога бота.

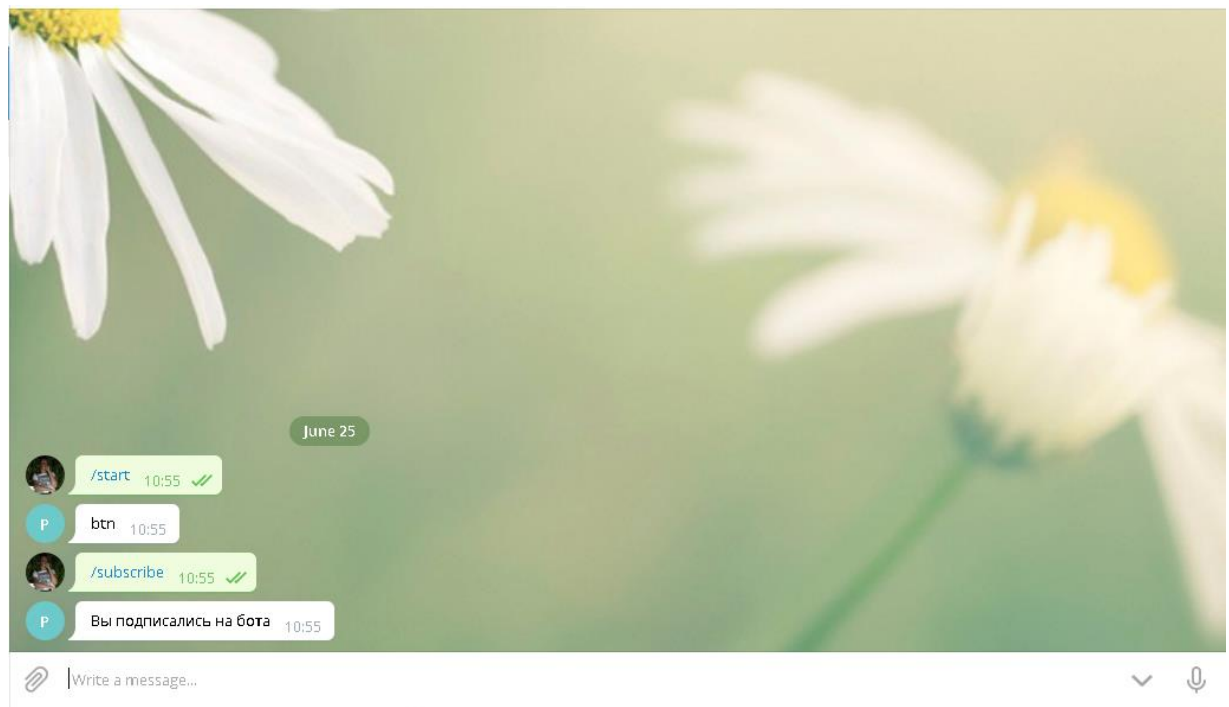


Рисунок 19. Результат работы подписки

### 3.2.4. Выполнение функции отписки от чат-бота

Нажмите на кнопку /unsubscribe находящуюся в окне диалога бота.

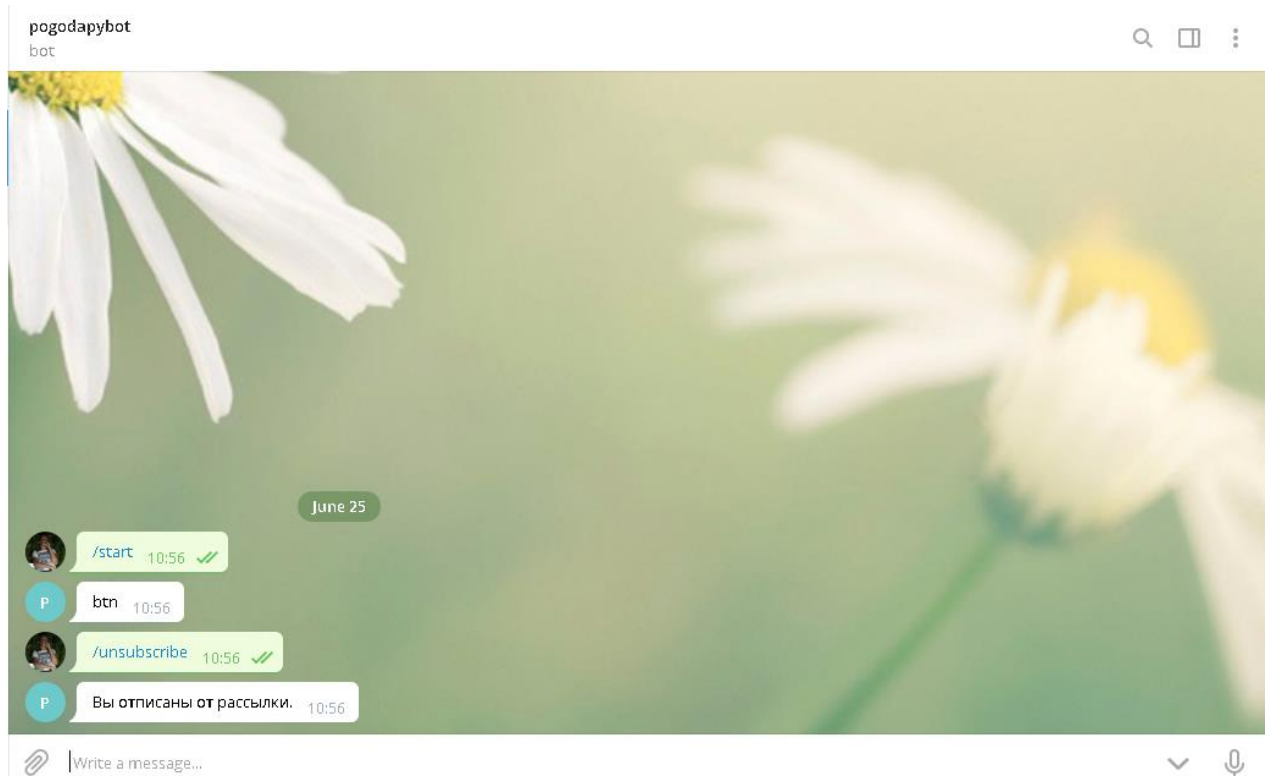


Рисунок 20. Результат работы отписки



### 3.2.5. Выполнение функции запроса погоды

Нажмите на кнопку `/weather` находящуюся в окне диалога бота и введите город для прогноза погоды

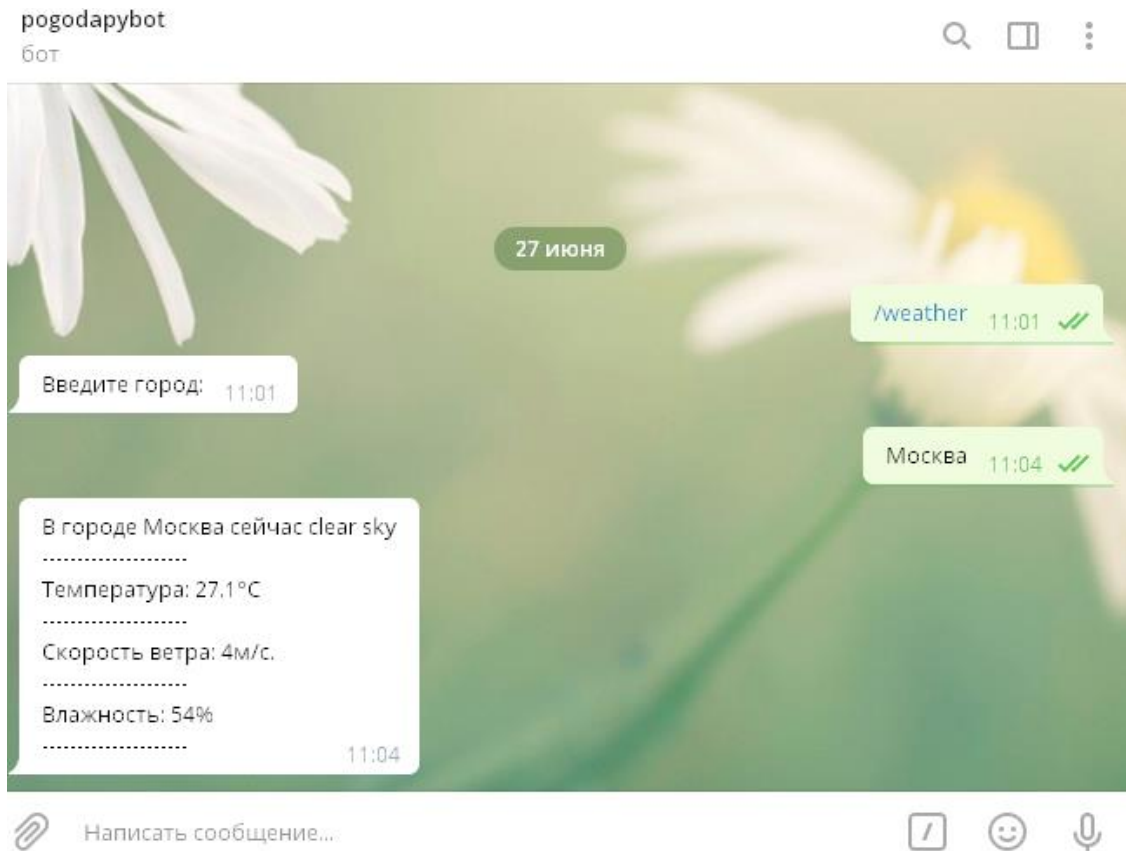


Рисунок 21. Прогноз погоды в городе

### 3.3. Завершение работы программы

Чтобы завершить работу программы нажмите на кнопку «Закреть» в верхнем правом углу окна.

## Заключение

В результате выполнения курсового проекта была написана программа «Pogoda bot» для просмотра прогноза погоды в городах мира.

В ходе работы были проанализированы предметная область, существующие разработки, посвященные данному направлению, получены практические навыки по работе с Aiogram API, Telegram Bot API, ruowm API и СУБД SQLite.

Также планируется продолжать работу над данным проектом с целью расширения возможностей и удобства приложения для пользователей. Планы по доработкам представлены ниже.

To-do лист:

1. Доработка прогноза погоды.
2. Добавления прогноза погоды на завтра, три и 5 дней вперед.
3. Доработка интерфейса с целью упрощения работы с программой.
4. Добавление рассылки новостей и прогноза погоды пользователям из списка подписчиков Базы Данных бота.
5. Расширение функционала чат-бота.

### Список литературы и интернет-источников

1. API для главного модуля программы:  
[https://docs.aiogram.dev/en/latest/\\_modules/aiogram/bot/api.html](https://docs.aiogram.dev/en/latest/_modules/aiogram/bot/api.html)
2. API для модуля погоды  
<https://pyowm.readthedocs.io/en/latest/>
3. Информация по работе с Telegram API:  
<https://core.telegram.org/>
4. API для модуля sqlite3  
<https://docs.python.org/2/library/sqlite3.html>
5. Статья про Async IO в Python:  
<https://webdevblog.ru/obzor-async-io-v-python-3-7/>

## Приложение 1. Код главного модуля bot.py.

```
# Булатников Илья П1-17
#импорт модулей для работы бота
import keyboard as kb
import logging
import asyncio
import pyowm
import geo
import aiogram.types.base

from aiogram import Bot, Dispatcher, executor, types
#импорт модулей для работы с TelegramBotAPI

from sqlighter import SQLighter #импорт класса для работы с БД

#задаем уровень логов
logging.basicConfig(level=logging.INFO)

#инициализация бота
bot = Bot(token='1296584769:AAEu7y4Qjz9y7k6zA0rn8Tz9PmzUnJA4mB0')

dp = Dispatcher(bot)

# инициализация соединения с БД
db = SQLighter('db.db')

#команда активации подписки
@dp.message_handler(commands=['subscribe'])
#если сообщение от пользователя 'subscribe' делать дальнейшие команды

#async def – определяет native coroutine function, результатом вызова
#которой будет объект-сопрограмма native coroutine, пока еще не запущенная.
async def subscribe(message: types.Message):

    #проверка на наличие подписки у пользователя
    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)

#async def – Сопрограмма (coroutine) асинхронная функция
async def subscribe(message: types.Message):

    #проверка на наличие подписки у пользователя
    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)
        #Ключевое слово await приостанавливает выполнение текущей
        #сопрограммы (coroutine) и вызывает message.answer.
        await message.answer("Вы подписались на бота")

#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):
        # сохраняем данные о городе
        city = message.text
        # печать в чат погоды
        await message.answer(geo.weatherd(city))

#вывод кнопки команд на экран
@dp.message_handler(commands=['start'])
async def buttons(message: types.Message):
    await message.answer('btn', reply_markup=kb.markup3)

#скрытие с экрана кнопок
@dp.message_handler(commands=['remove'])
async def removekb(message: types.Message):
    await message.answer('remove_kb', reply_markup=kb.ReplyKeyboardRemove())

#запуск бота
if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)

```

## Приложение 2. Код модуля экранной клавиатуры keybord.py.

```
# Звонарев Данила П1-17
#импорт модулей для работы с клавиатурой и кнопками
from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup,
ReplyKeyboardRemove, KeyboardButton, ReplyKeyboardMarkup

#кнопки для меню
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')

inline_kb1 = InlineKeyboardMarkup()
#печать в чат поля 3*3 из кнопок для будущего меню
inline_kb1.add(
    button1, button2, button3, button4, button5, button6, button7, button8,
    button9
)
#кнопки для виртуальной клавиатуры
button10 = KeyboardButton('/weather')
button11 = KeyboardButton('/subscribe')
button12 = KeyboardButton('/unsubscribe')
#вывод на экран виртуальной клавиатуры с кнопками
markup3 = ReplyKeyboardMarkup().row(
    button10, button11, button12
)
```

### Приложение 3. Код модуля базы данных sqlighter.py.

```
# Булатников Илья П1-17
#импорт модуля для работы с БД
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()
```

## Приложение 4. Код модуля работы с погодой гео.ру.

```
# Звонарев Данила П1-17
import pyowm
#импорт модуля для работы с погодой

#погода
def weatherd(city):

    # инициализируем бота для работы с погодой
    owm = pyowm.OWM('a7a5f151b3845f1d0a5979f764dbb267')
    mgr = owm.weather_manager()
    # получение информации о погоде в городе
    observation = mgr.weather_at_place(city + ',rus')
    # погода
    w = observation.weather
    # температура
    temp = w.temperature('celsius')['temp']
    # скорость ветра
    wind = w.wind()['speed']
    # погода влажность
    hum = w.humidity
    # формирование ответа, выводимого в чат пользователю
    answer = "В городе " + city + " сейчас " + w.detailed_status + "\n-----
-----"
    answer += "\nТемпература: " + str(temp) + "°C" + "\n-----"
    " + "\nСкорость ветра: " + str(wind) + "м/с."
    answer += "\n-----" + "\nВлажность: " + str(hum) + "%" +
    "\n-----"
    return answer
```