

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](#_Toc44148915)

[**1.** **Теоретическая часть** 4](#_Toc44148916)

[**1.1.** **Описание предметной области** 4](#_Toc44148917)

[**1.2.** **О Ботах** 6](#_Toc44148918)

[**1.3.** **Описание существующих разработок** 8](#_Toc44148919)

[**1.3.1.** **Бот “Шеф-повар”** 9](#_Toc44148920)

[**1.3.2.** **Бот “Транспорт Минск”** 10](#_Toc44148921)

[**2.** **Проектная часть** 11](#_Toc44148922)

[**2.1.** **Диаграмма прецедентов** 11](#_Toc44148923)

[**2.2.** **Выбор инструментов** 12](#_Toc44148924)

[**2.3.** **Проектирование сценария** 13](#_Toc44148925)

[**2.4.** **Диаграмма классов** 14](#_Toc44148926)

[**2.5.** **Описание главного модуля** 15](#_Toc44148927)

[**2.6.** **Описание спецификаций к модулям** 17](#_Toc44148928)

[**2.7.** **Описание спецификаций к модулям** 18](#_Toc44148929)

[**2.7.1.** **Keybord.py** 18](#_Toc44148930)

[**2.7.2.** **Geo.py** 19](#_Toc44148931)

[**2.7.3.** **sqlighter.py** 20](#_Toc44148932)

[**2.8.** **Описание тестовых наборов модулей** 21](#_Toc44148933)

[**2.9.** **Описание применения средств отладки** 24](#_Toc44148934)

[**2.10.** **Анализ оптимальности использования памяти и быстродействия** 26](#_Toc44148935)

[**3.** **Эксплуатационная часть** 27](#_Toc44148936)

[**3.1.** **Руководство оператора** 27](#_Toc44148937)

[**Заключение** 34](#_Toc44148938)

[**Список литературы и интернет-источников** 35](#_Toc44148939)

[**Приложение 1. Код главного модуля bot.py.** 36](#_Toc44148940)

[**Приложение 2. Код модуля экранной клавиатуры keybord.py.** 38](#_Toc44148941)

[**Приложение 3. Код модуля базы данных sqlighter.py.** 39](#_Toc44148942)

[**Приложение 4. Код модуля работы с погодой geo.py.** 40](#_Toc44148943)

# **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.

# **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.

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 c 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.

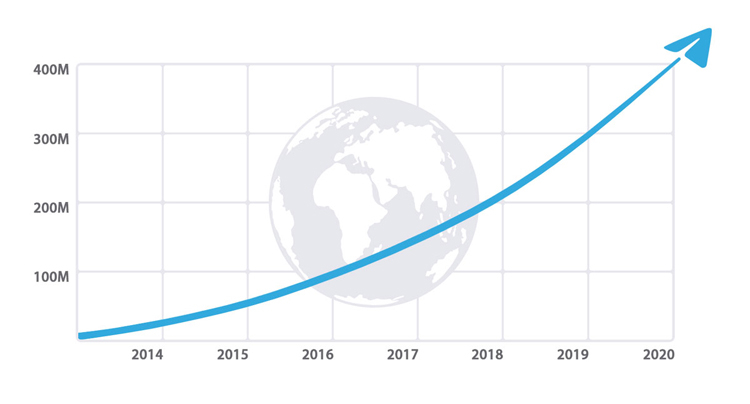


Рисунок 1. График роста аудитории Telegram

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.

## **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 сокращении 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-бот **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.

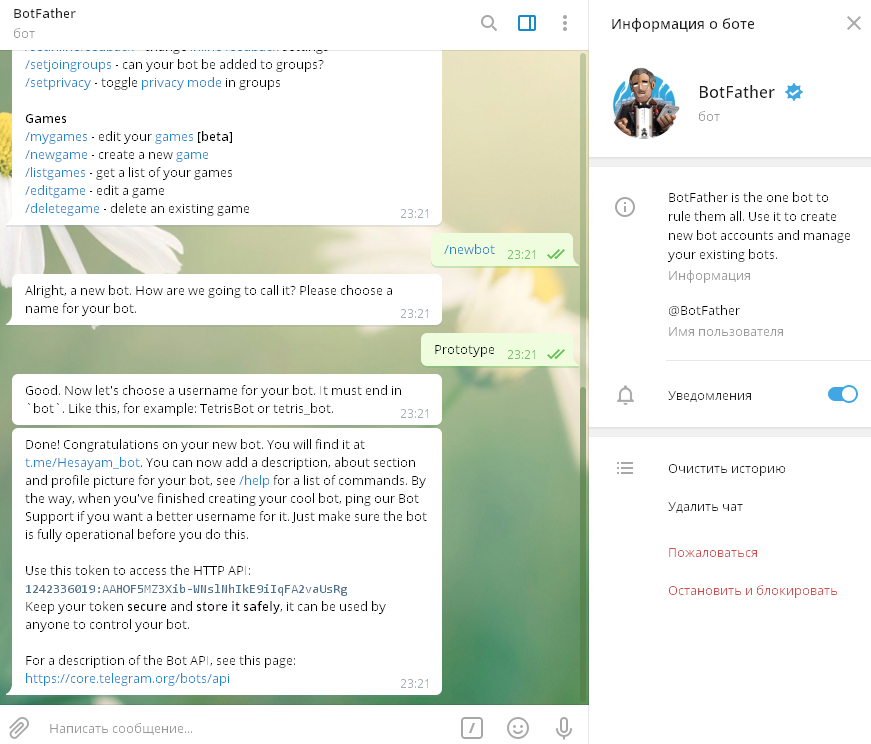


Рисунок 2. Создание бота в Telegram

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.

## **Описание существующих разработок**

В это 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.

## **Бот “Шеф-повар”**

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

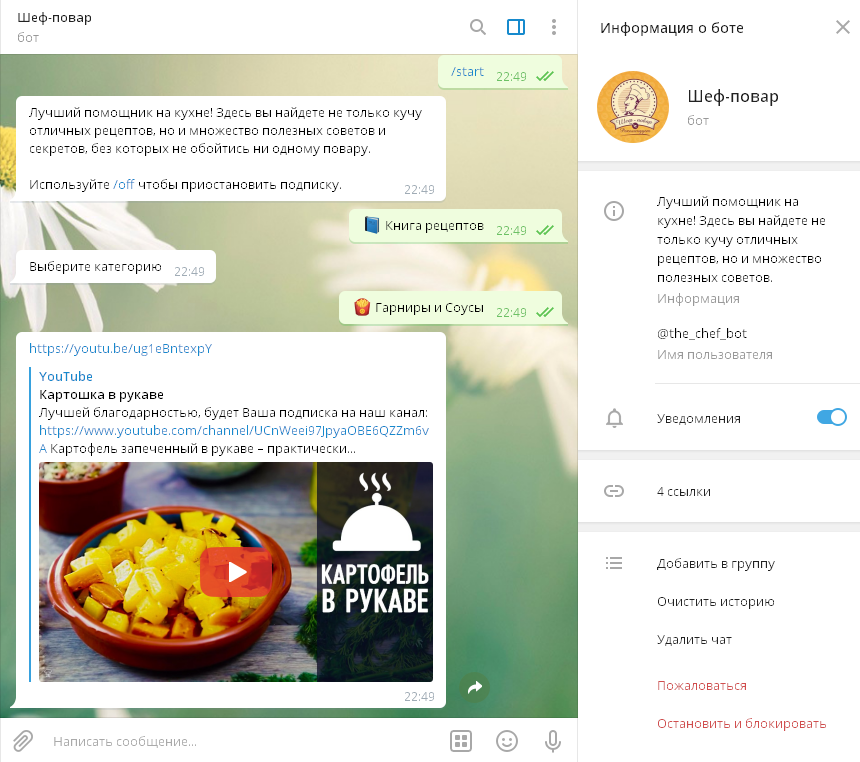


Рисунок 3. Бот Шеф-повар

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

## **Бот “Транспорт Минск”**

Этот чат-бот предназначен для просмотра расписание автобусов

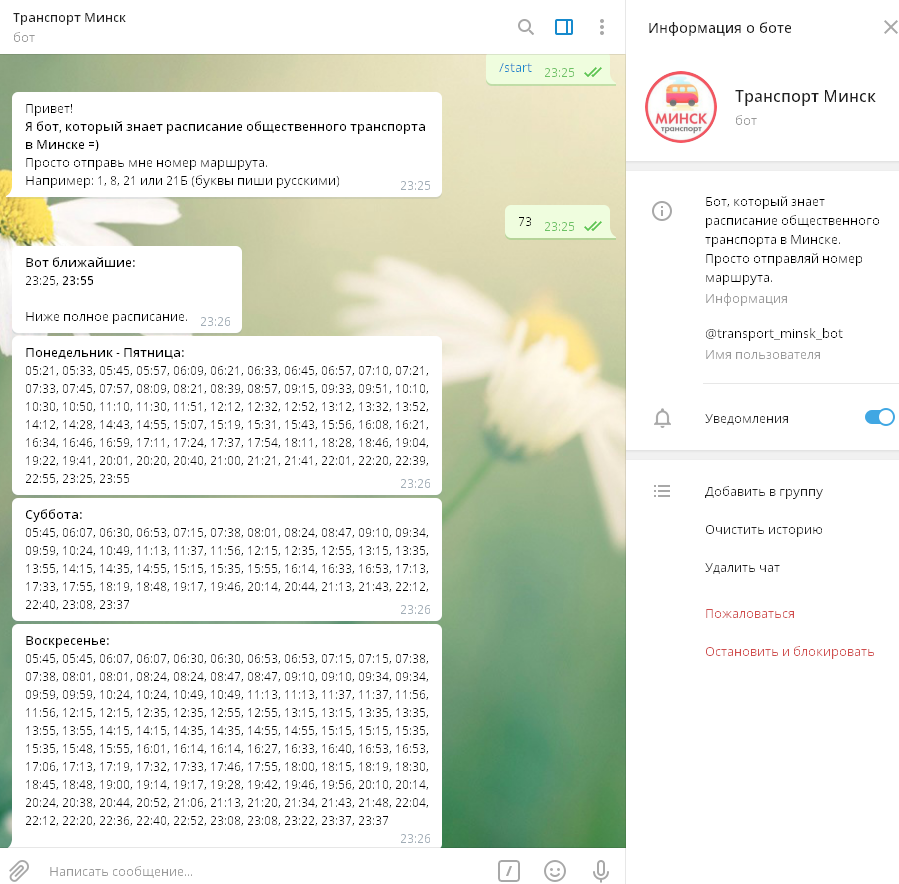
**

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.

# **Проектная часть**

## **Диаграмма прецедентов**

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

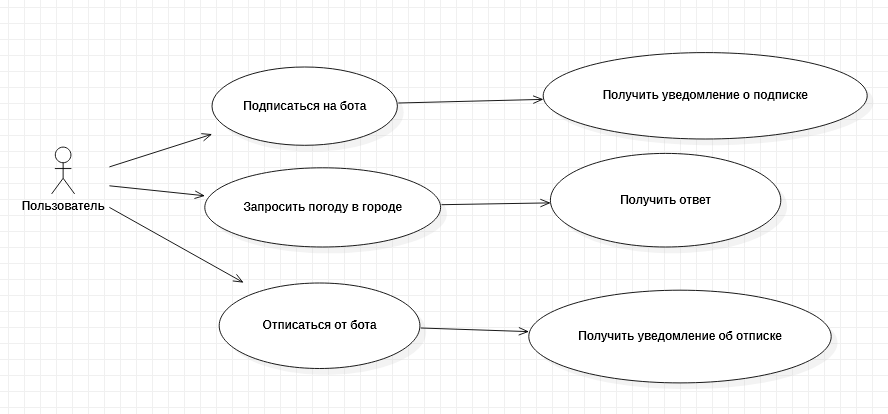


Рисунок 5. Диаграмма "Пользователь"

Взаимодействие с чат-ботом ведется посредством виртуальной клавиатуры.

## **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.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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>

## **Проектирование сценария**

В данном разделе приведен сценарий использования программы пользователем.

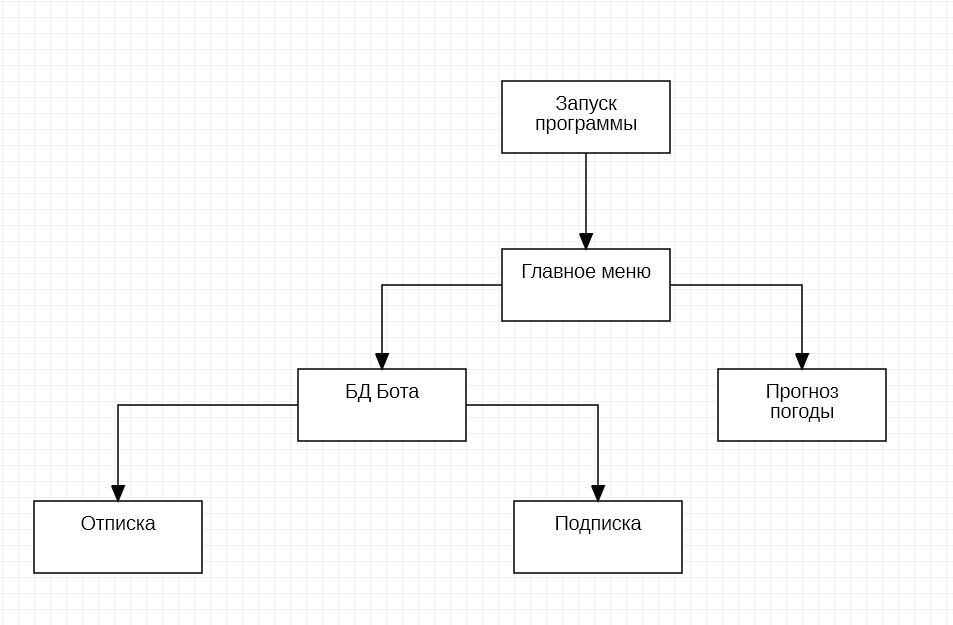


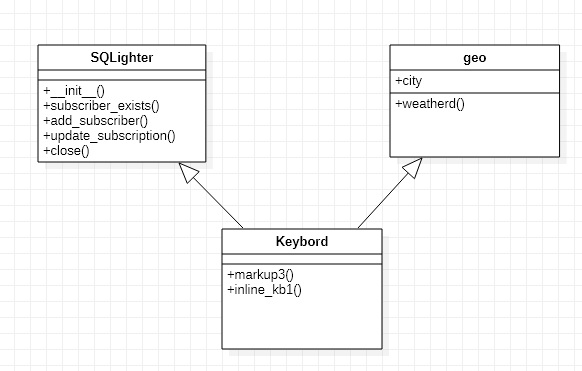
Рисунок 6. Сценарий использования

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.

## **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

## **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.**

#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):

city = message.text # сохраняем данные о городе

await message.answer(geo.weatherd(city)) # печать в чат погоды

#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.

## **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.

## **Описание модулей**

Кроме главного модуля программа содержит три модуля.

Они расширяют функционал основного модуля.

## **Keybord.py**

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

**Листинг 3. Модуль клавиатуры и меню.**

#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

## **Geo.py**

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

**Листинг 4. Модуль погоды.**

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." + "\n---------------------" + "\nВлажность: " + str(hum) + "%" + "\n---------------------"

return answer

## **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 содержаться в класс SQLighter из модуля sqlighter .py.

**Листинг 5. Методы класса 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()

## **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:

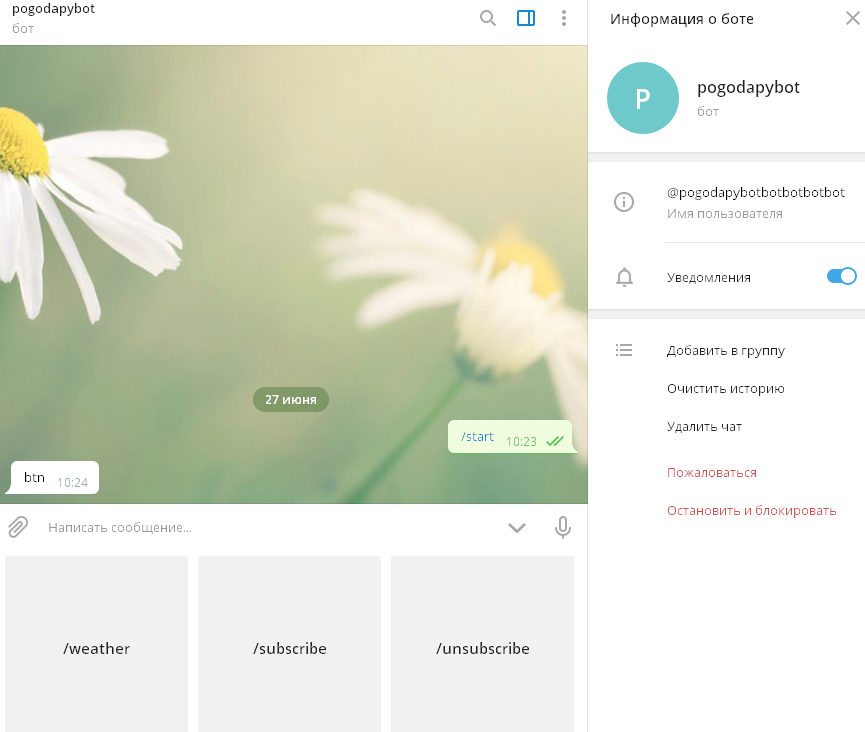


Рисунок 9. Экранная клавиатура

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:

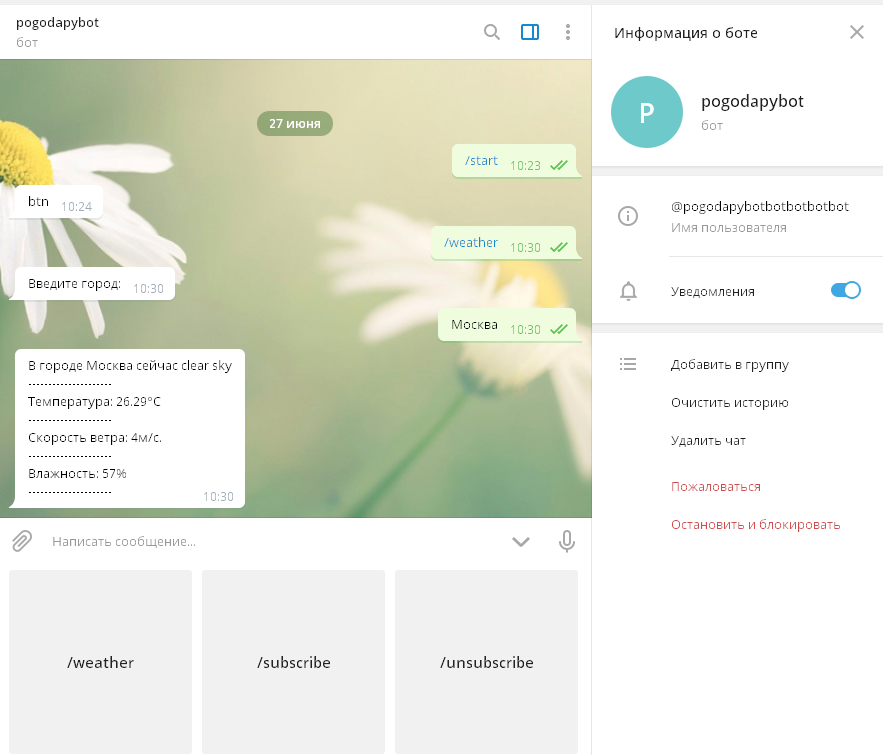


Рисунок 10. Погода в Москве

Тест 3. Проверка подписки.

Действия: Нажать на кнопку “/subscribe”.

Ожидаемый результат: Сообщение о подписке.

Результат теста:

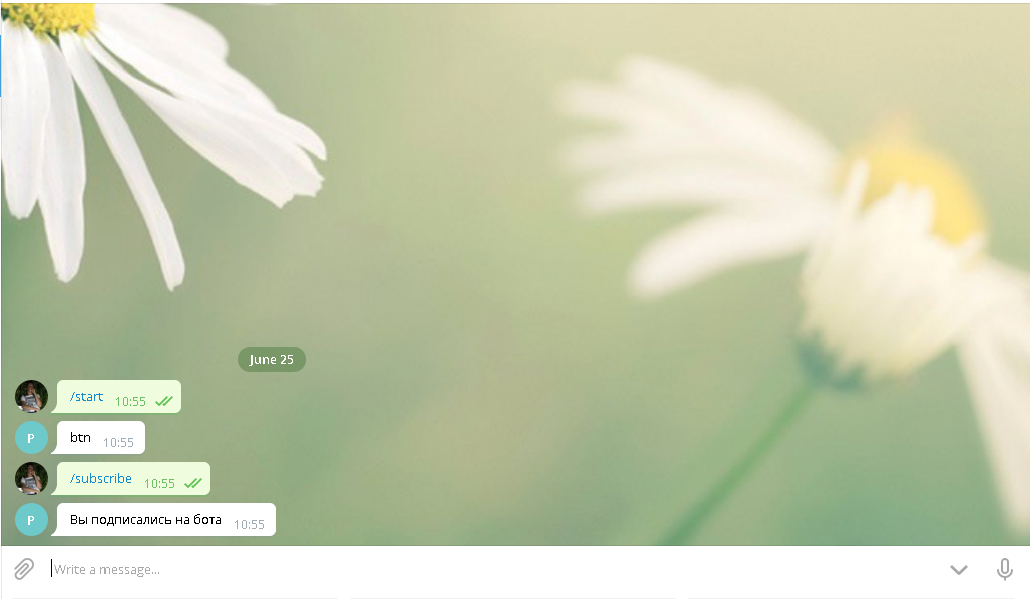


Рисунок 11. Уведомление о подписке

Тест 4. Проверка отписки.

Действия: Нажать на кнопку “/unsubscribe”.

Ожидаемый результат: Сообщение об отписке.

Результат теста:

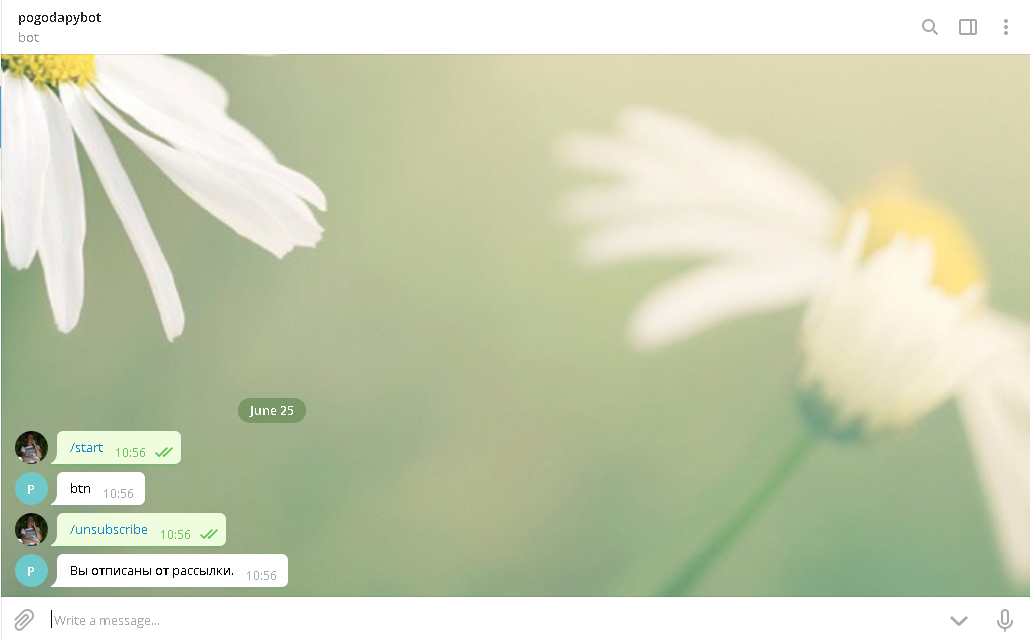


Рисунок 12. Уведомление о подписке

## **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:

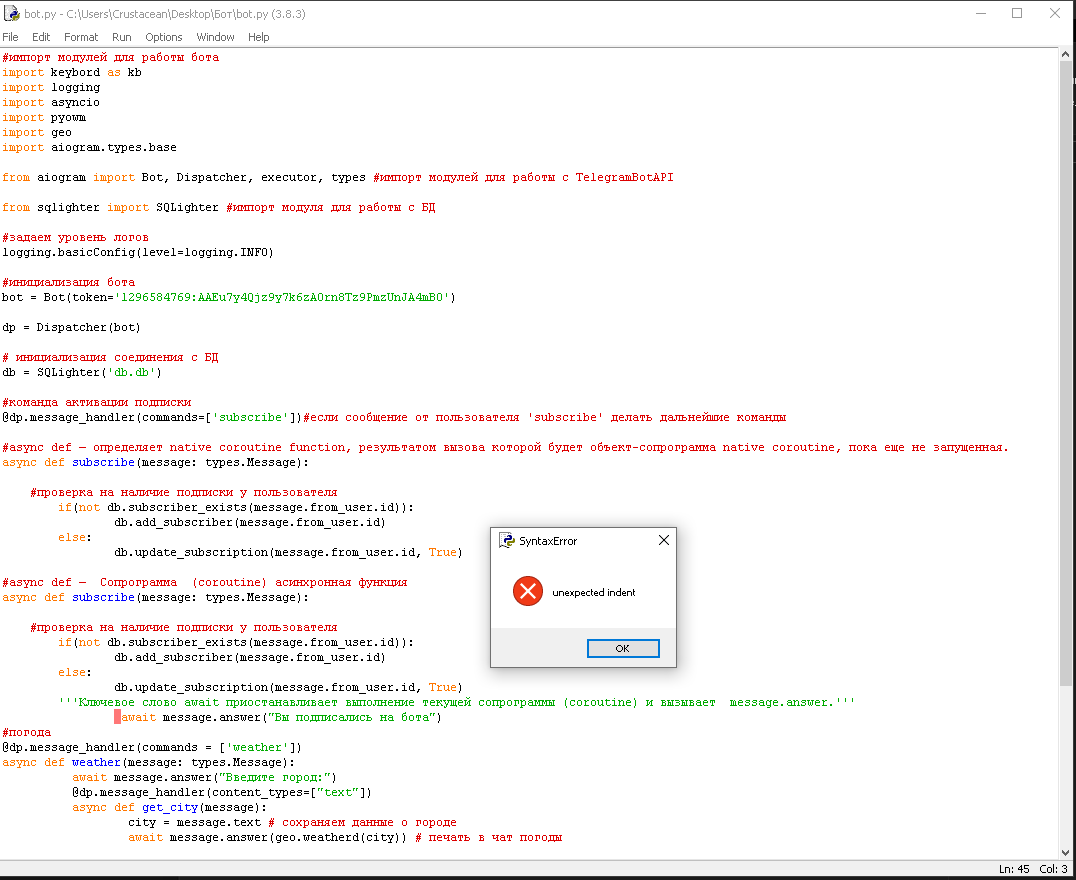


Рисунок 13. До применения средств отладки

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:

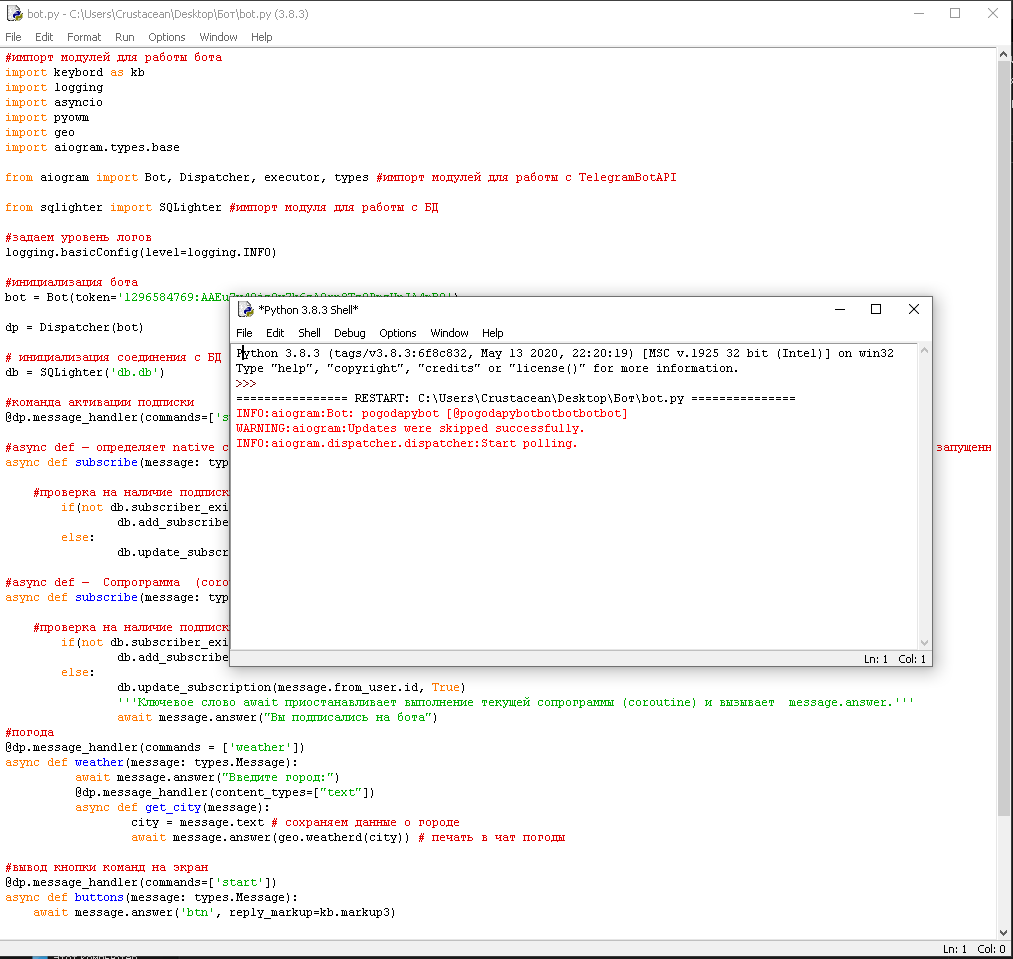


Рисунок 14. После применения средств отладки

Это означает что ошибка была устранена и скрипт запустился.

## **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.

1. 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

# **Эксплуатационная часть**

## **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 [[1]](#footnote-1)1), ГОСТ 19.103-77 [[2]](#footnote-2)2), ГОСТ 19.104-78\* [[3]](#footnote-3)3), ГОСТ 19.105-78\* [[4]](#footnote-4)4), ГОСТ 19.106-78\* [[5]](#footnote-5)5), ГОСТ 19.505-79\* [[6]](#footnote-6)6), ГОСТ 19.604-78\* [[7]](#footnote-7)7)).

1. **Назначение программы**
   1. **Функциональное назначение программы**

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

* 1. **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. **Состав функций**
     1. **Функция открытия меню.**

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

* + 1. **Функция подписки на чат-бота**

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

* + 1. **Функция отписки от чат-бота.**

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

* + 1. **Функция запроса погоды**

Эта функция позволяет узнать текущую погоду в введенном городе.

1. **Условия выполнения программы**
   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.

* 1. **Минимальный состав программных средств**

Telegram messenger

SQLite3

* 1. **Требование к персоналу (пользователю)**

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

1. **Выполнение программы**
   1. **Загрузка и запуск программы**

Перед запуском программы необходимо установить:

* СУБД SQLite
* Telegram

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

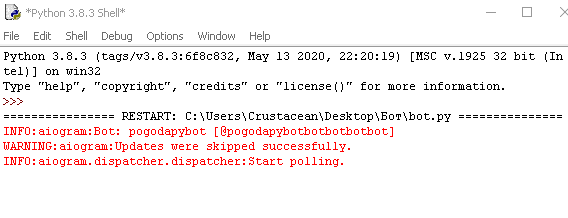


Рисунок 15. Уведомление о запуске бота

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

* 1. **Выполнение программы**
     1. **Поиск чат-бота в списке каналов.**

Запустите Telegram и в поиске каналов введите “@pogodapybotbotbotbotbot”

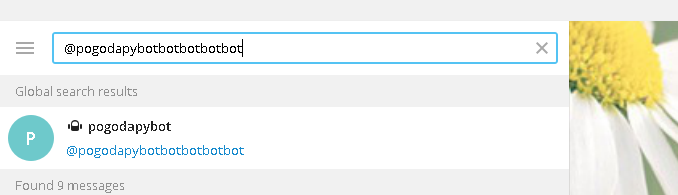


Рисунок 16. Поиск бота в списке каналов

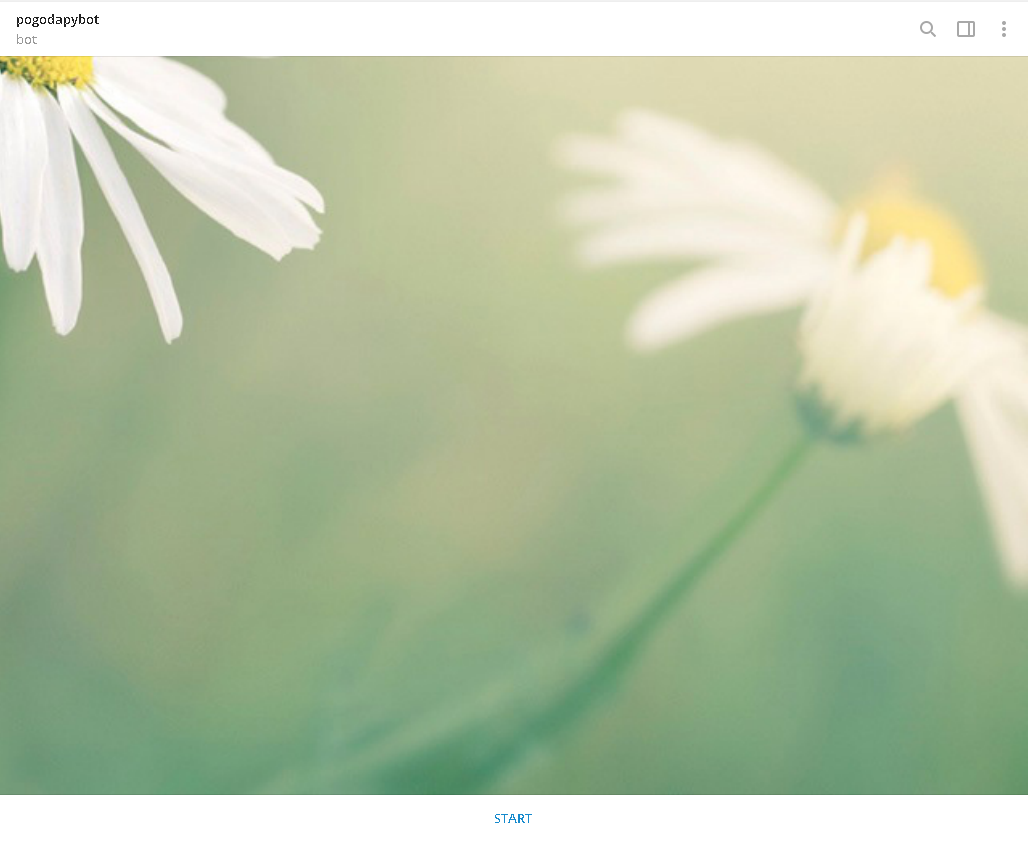


Рисунок 17. Начальное окно диалога чат-бота

* + 1. **Выполнение функции открытия меню.**

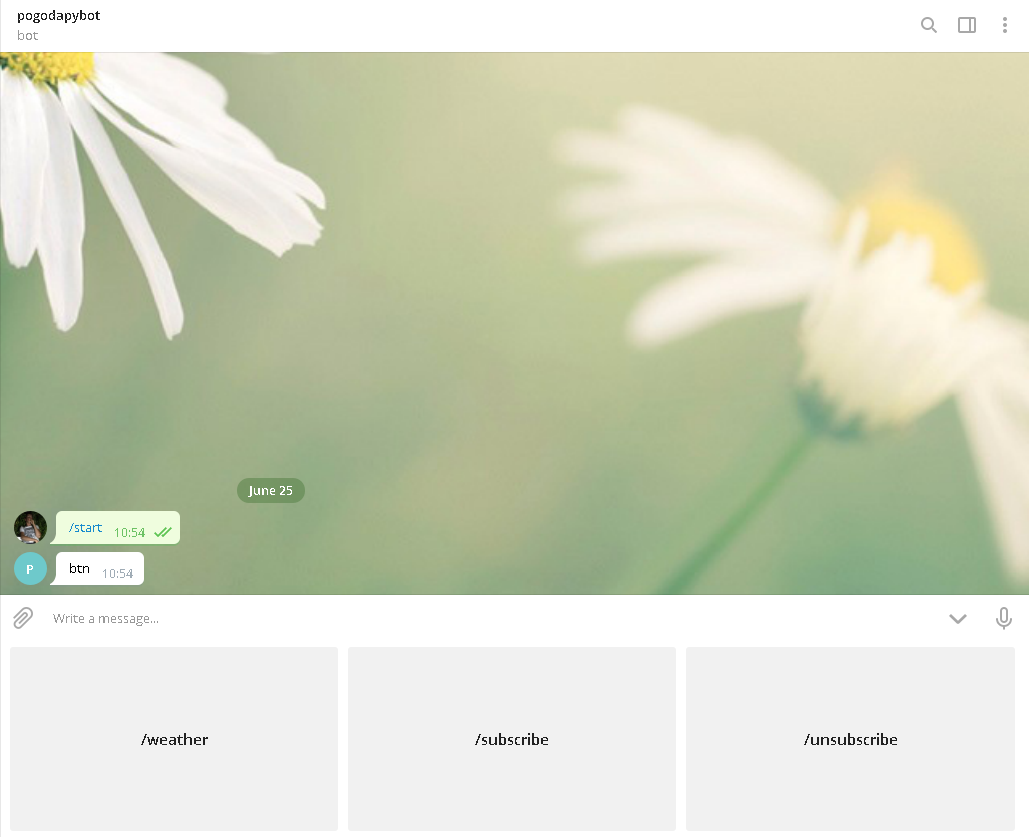
Нажмите на кнопку Start находящее в окне диалога чат-бота.

Рисунок 18. Экранная клавиатура с доступными командами

* + 1. **Выполнение функции подписки на чат-бота.**

Нажмите на кнопку /subscribe находящее в окне диалога бота.

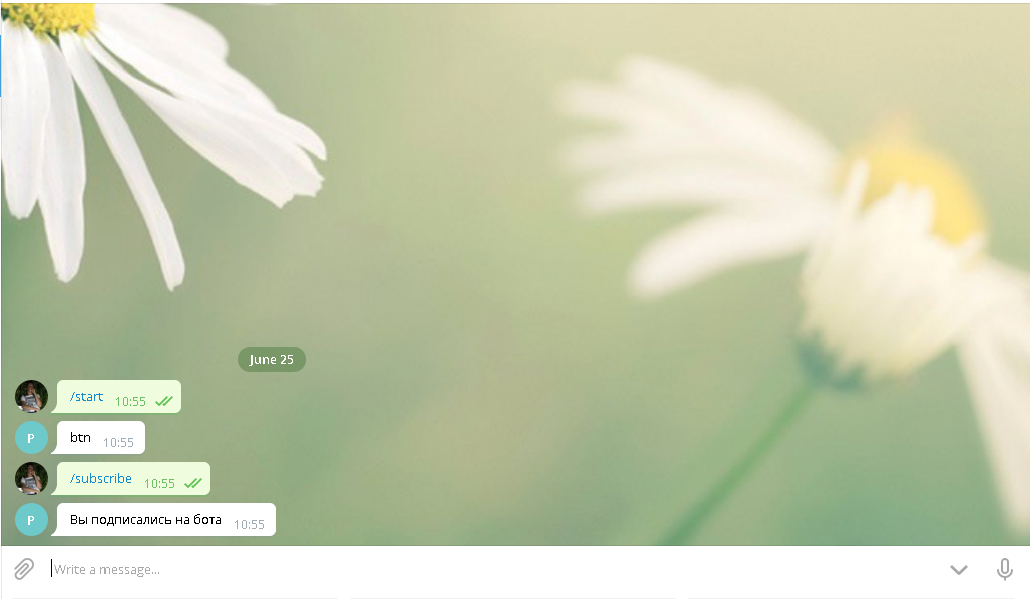


Рисунок 19. Результат работы подписки

* + 1. **Выполнение функции отписки от чат-бота**

Нажмите на кнопку /unsubscribe находящее в окне диалога бота.

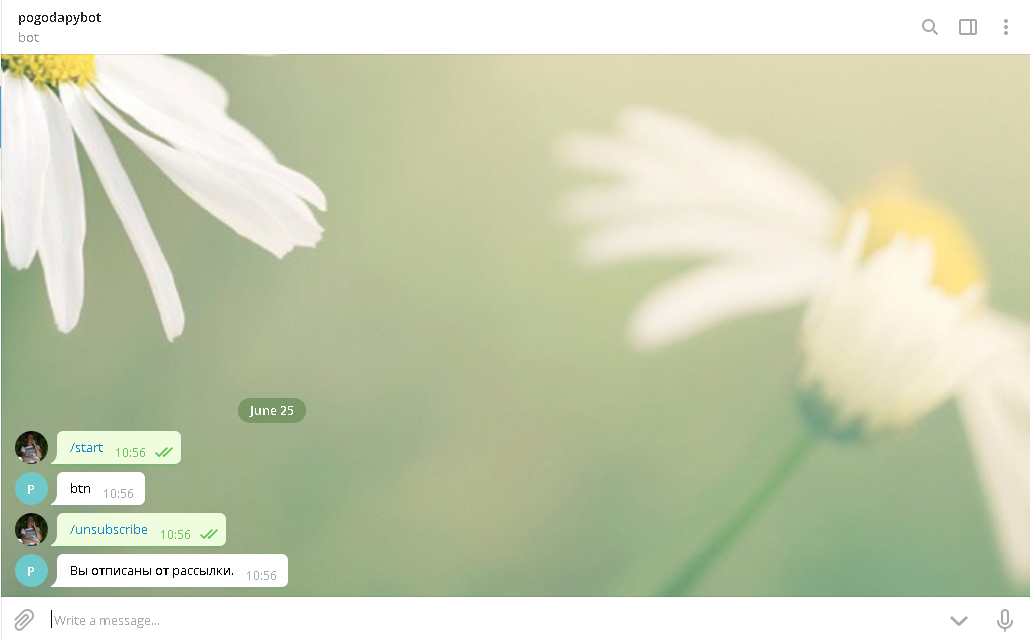


Рисунок 20. Результат работы отписки

* + 1. **Выполнение функции запроса погоды**

Нажмите на кнопку /weather находящее в окне диалога бота и ведите город для прогноза погоды

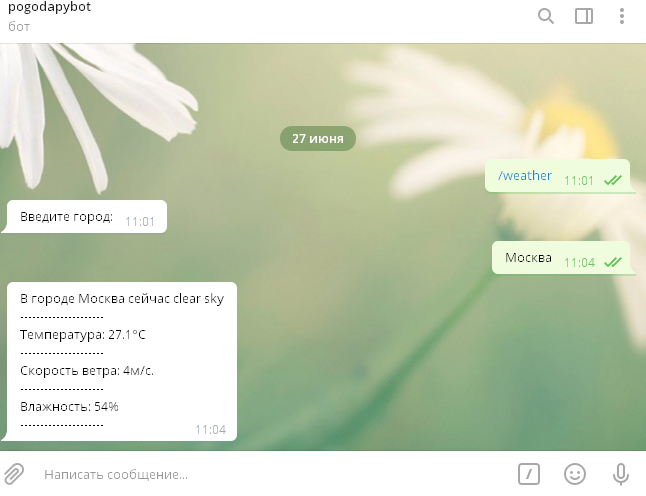


Рисунок 21. Прогноз погоды в городе

* 1. **Завершение работы программы**

Чтобы завершить работу программы нажмите на кнопку «Закрыть» 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, pyowm 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 вперёд.
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>

1. API для модуля погоды

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

1. Информация по работе с Telegram API:

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

1. API для модуля sqlite3

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

1. Статья про Async IO в Python:

https://webdevblog.ru/obzor-async-io-v-python-3-7/

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

#импорт модулей для работы бота

import keybord 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 модуля 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):

city = message.text # сохраняем данные о городе

await message.answer(geo.weatherd(city)) # печать в чат погоды

#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.**

#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.**

import sqlite3 #импорт модуля для работы с БД

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 geo.py.**

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." + "\n---------------------" + "\nВлажность: " + str(hum) + "%" + "\n---------------------"

return answer

1. 1) ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов [↑](#footnote-ref-1)
2. 2) ГОСТ 19.103-77 ЕСПД. Обозначение программ и программных документов [↑](#footnote-ref-2)
3. 3) ГОСТ 19.104-78\* ЕСПД. Основные надписи [↑](#footnote-ref-3)
4. 4) ГОСТ 19.105-78\* ЕСПД. Общие требования к программным документам [↑](#footnote-ref-4)
5. 5) ГОСТ 19.106-78\* ЕСПД. Общие требования к программным документам, выполненным печатным способом [↑](#footnote-ref-5)
6. 6) ГОСТ 19.505-79\* ЕСПД. Руководство оператора. Требования к содержанию и оформлению [↑](#footnote-ref-6)
7. 7) ГОСТ 19.604-78\* ЕСПД. Правила внесения изменений в программные документы, выполненные печатным способом [↑](#footnote-ref-7)