

Колледж космического машиностроения и технологий

КУРСОВОЙ ПРОЕКТ

По МДК.01.02 «Прикладное программирование»

Тема: «Разработка бот telegram: главный модуль»

Выполнил студент

Булатников Илья Сергеевич

Группа П1-17

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

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

Проверил преподаватель

Гусятинер Леонид Борисович

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Оценка)

Королёв 2020 г.

Оглавление

[Введение 2](#_Toc44162654)

[Глава 1. Теоретическая часть 3](#_Toc44162655)

[1.1. Описание предметной области 3](#_Toc44162656)

[1.2. Описание существующих разработок 9](#_Toc44162657)

[Глава 2. Проектная часть 11](#_Toc44162658)

[2.1. Диаграмма прецедентов 11](#_Toc44162659)

[2.2. Выбор инструментов 12](#_Toc44162660)

[2.3. Проектирование сценария 13](#_Toc44162661)

[2.4. Диаграмма классов 14](#_Toc44162662)

[2.5. Описание главного модуля 15](#_Toc44162663)

[2.6. Описание спецификаций к модулям 17](#_Toc44162664)

[2.7. Описание модулей 18](#_Toc44162665)

[2.8. Описание тестовых наборов модулей 21](#_Toc44162666)

[2.9. Описание применения средств отладки 24](#_Toc44162667)

[2.10. Анализ оптимальности использования памяти и быстродействия 25](#_Toc44162668)

[Глава 3. Эксплуатационная часть 26](#_Toc44162669)

[Список литературы и интернет-источников 33](#_Toc44162670)

[Приложение 1. Код главного модуля. 34](#_Toc44162671)

[Приложение 2. Модуль виртуальной клавиатуры 36](#_Toc44162672)

[Приложение 3. Модуль погоды 37](#_Toc44162673)

[Приложение 4. Модуль БД 38](#_Toc44162674)

## 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

## 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 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.

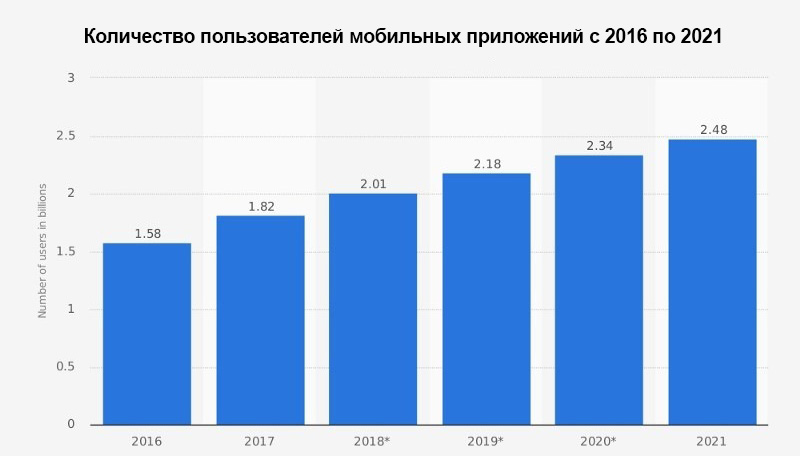


Рисунок 1.Диаграмма роста пользователей на мобильном рынке

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.

Использовать для написания мы будем мессенджер Telegram, поэтому поподробнее о нем.

Telegram — уникальный по своей структуре мессенджер, являющийся кроссплатформенным приложением. Помимо стандартного обмена сообщениями в диалогах и группах, в мессенджере можно хранить неограниченное количество файлов, вести каналы (микроблоги), создавать и использовать ботов. При помощи специального API сторонние разработчики могут создавать боты, специальные аккаунты, управляемые программами. Типичные боты отвечают на специальные команды в персональных и групповых чатах, также они могут осуществлять поиск в интернете или выполнять иные задачи, применяются в развлекательных целях или в бизнесе [29]. Также реализована функция использования socks5 proxy, которая позволяет использовать мессенджер даже в тяжелых для подключения условиях. SOCKS 4/5 — это распространённое обозначение двух версий сетевого протокола SOCKS 4 и SOCKS 5. Основным преимуществом SOCKS (от. англ. 18 «sockets» — гнёзда) является возможность работы клиент-серверных приложений за границами межсетевого экранирования, то есть SOCKS прокси может принять запрос от клиента, находящегося за фаерволом, просмотреть его права доступа и передать запрос на внешний сервер. Telegram-боты — разновидность чат-ботов. Их суть заключается в ответной реакции на определенные сообщения от пользователей. Таким образом, сфера их применения безгранична. Компания Uber, некоторые банки, крупные магазины и многие другие организации используют Telegram-ботов для упрощения и автоматизации внутренних рабочих процессов. При регистрации бота выдается уникальный ключ, с помощью которого в дальнейшем и будет происходить связь между клиентом и сервером. Такая схема исключает необходимость дополнительных настроек клиент-серверной архитектуры, так как все происходит автоматически и занимает несколько строчек кода. По вышеуказанным причинам Telegram был выбран как самая удобная оболочка для разрабатываемого приложения.

Боты — особые аккаунты, которые не требуют номера телефона при создании. По сути, эти аккаунты играют роль интерфейса к вашему сервису, который работает на удалённом сервере.

При помощи специального API сторонние разработчики могут создавать «ботов», специальные аккаунты, управляемые программами. Типичные боты отвечают на специальные команды в персональных и групповых чатах, также они могут осуществлять поиск в интернете или выполнять иные задачи, применяются в развлекательных целях или в бизнесе.

В Telegram есть несколько видов ботов:

* Чат-боты. Представляют из себя простейший чат, имитирующий общение на заданную пользователем тематику.
* Боты-информаторы. Отдельный вид ботов, главная цель которых — информирование пользователя о тех или иных событиях (новости, мероприятия, публикации и т.п.).
* Игровые боты. Боты, в которых можно поиграть в различные игры. По большей части, это текстовые версии разных игр времен Combats.ru
* Боты-асистенты. Боты, разработанные различными онлайн-сервисами как дополнение к основной веб-версии.

И писать мы будем чат-бота

Такого бота можно использовать для рассылки сообщений или сбора информации.

Этот модуль является оболочкой над запросами к Telegram Bot API, используется для упрощения и минимизации написанного кода. Все типы определены в types.py. Все они полностью соответствуют определению типов API Telegram, за исключением from поля Message, которое переименовано в from\_user(поскольку from это зарезервированный токен Python). Таким образом, к таким атрибутам как message\_id можно обращаться напрямую, например: message.message\_id. Стоит обратить внимание, что атрибут message.chat может принадлежать как определенному пользователю, так и групповому чату. 6 Объект Message также имеет content\_type атрибут, который определяет тип сообщения. Атрибут content\_type может быть одним из следующих строк: text, audio, document, photo, sticker, video и так далее. Можно использовать несколько типов в одной функции.

Все методы API расположены в классе TeleBot. Они переименованы, чтобы следовать общим соглашениям об именах Python. Например: sendMessage переименован в send\_message, editMessageText в edit\_message\_text. Обработчик сообщений - это функция, украшенная декоратором экземпляра TeleBot. Обработчики сообщений состоят из одного или нескольких фильтров. Каждый фильтр возвращает True или False для определенного сообщения, и если возвращается True, то обработчик получает разрешение на обработку сообщения.

В документации Telegram Bot API выделяется два диаметрально противоположных способа получения обновлений: c помощью периодических запросов или установка вебхуков. Входящие обновления хранятся до тех пор, пока сервер не обработает его, но не дольше 24 часов. Независимо от способа получения обновлений, в ответ получаем объект Update, сериализованный в JSON. 13 Первый и наиболее простой вариант заключается в периодическом опросе серверов Telegram на предмет наличия новой информации. Всѐ это осуществляется через т.н. Long Polling, т.е. открывается соединение на непродолжительное время и все обновления тут же отправляются боту. Просто, но не очень надежно. Во-первых, серверы Telegram периодически начинают возвращать ошибку 504 (Gateway Timeout), из-за чего некоторые боты впадают в ступор. Во-вторых, если одновременно запущено несколько ботов, вероятность столкнуться с ошибками возрастает. Вебхуки работают несколько иначе. Под установкой вебхука подразумевается, что теперь если в чат приходит сообщение, то Telegram сам говорит об этом. Отпадает необходимость периодически опрашивать серверы, тем самым, исчезает причина падений ботов. Однако за это приходится платить необходимостью установки полноценного веб-сервера на ту машину, на которой планируется запускать ботов. Так же для работы надо иметь собственный SSL-сертификат (Secure Sockets Layer), т.к. вебхуки в Telegram работают только по HTTPS [12]. Для работы с Telegram Bot API была изучена документация, в которой описаны все методы и передаваемые параметры, все ответы приходят в JSON-формате. В ходе написания чат-бота были протестированы и использованы следующие методы и типы: метод getUpdates используется для получения обновлений через long polling. Ответ возвращается в виде массива объектов Update.

В качестве РСУБД мой выбор пал на SQLite, поэтому поподробнее о ней.

SQLite — это изумительная библиотека, встраиваемая в приложение, которое её использует. Будучи файловой БД, она предоставляет отличный набор инструментов для более простой (в сравнении с серверными БД) обработки любых видов данных.

Когда приложение использует SQLite, их связь производится с помощью функциональных и прямых вызовов файлов, содержащих данные (например, баз данных SQLite), а не какого-то интерфейса, что повышает скорость и производительность операций.

#### Преимущества

* **Файловая:** вся база данных хранится в одном файле, что облегчает перемещение.
* **Стандартизированная:** SQLite использует SQL; некоторые функции опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), однако, есть и некоторые новые.
* **Отлично подходит для разработки и даже тестирования:** во время этапа разработки большинству требуется масштабируемое решение. SQLite, со своим богатым набором функций, может предоставить более чем достаточный функционал.

#### Недостатки

* **Отсутствие пользовательского управления:** продвинутые БД предоставляют пользователям возможность управлять связями в таблицах в соответствии с привилегиями, но у SQLite такой функции нет.
* **Невозможность дополнительной настройки:** опять-таки, SQLite нельзя сделать более производительной, поковырявшись в настройках — так уж она устроена.

#### Поддерживаемые типы данных

* **NULL:** NULL-значение.
* **INTEGER:** целое со знаком, хранящееся в 1, 2, 3, 4, 6, или 8 байтах.
* **REAL:** число с плавающей запятой, хранящееся в 8-байтовом формате IEEE.
* **TEXT:** текстовая строка с кодировкой UTF-8, UTF-16BE или UTF-16LE.
* **BLOB:** тип данных, хранящийся точно в таком же виде, в каком и был получен.

**Note:** для получения более подробной информации ознакомьтесь с [документацией](https://www.sqlite.org/datatype3.html).

## 1.2. Описание существующих разработок

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

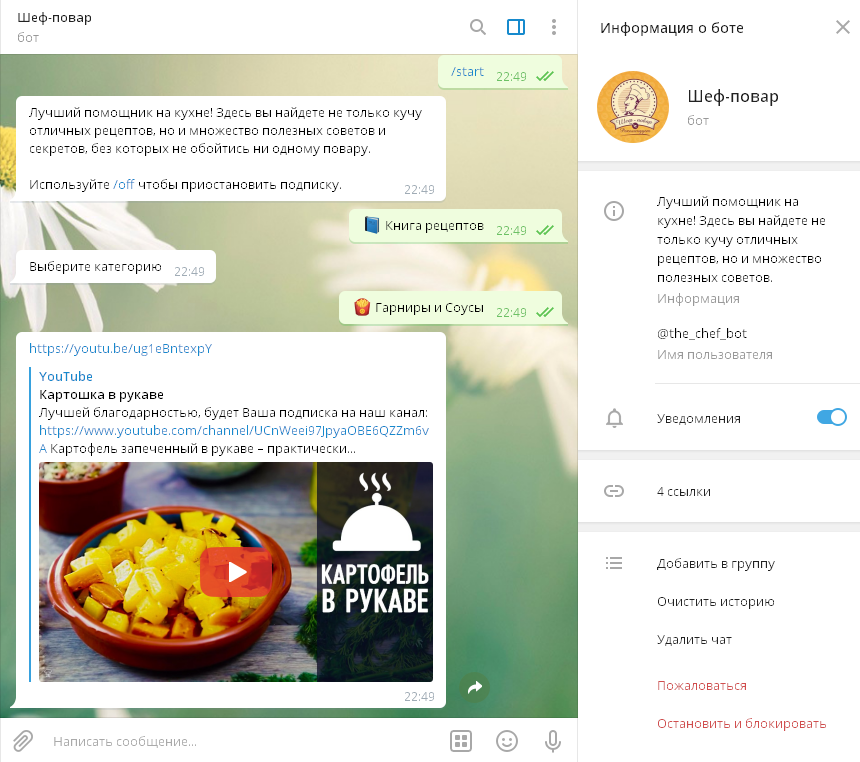


Рисунок 2.Бот Шеф-повар

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

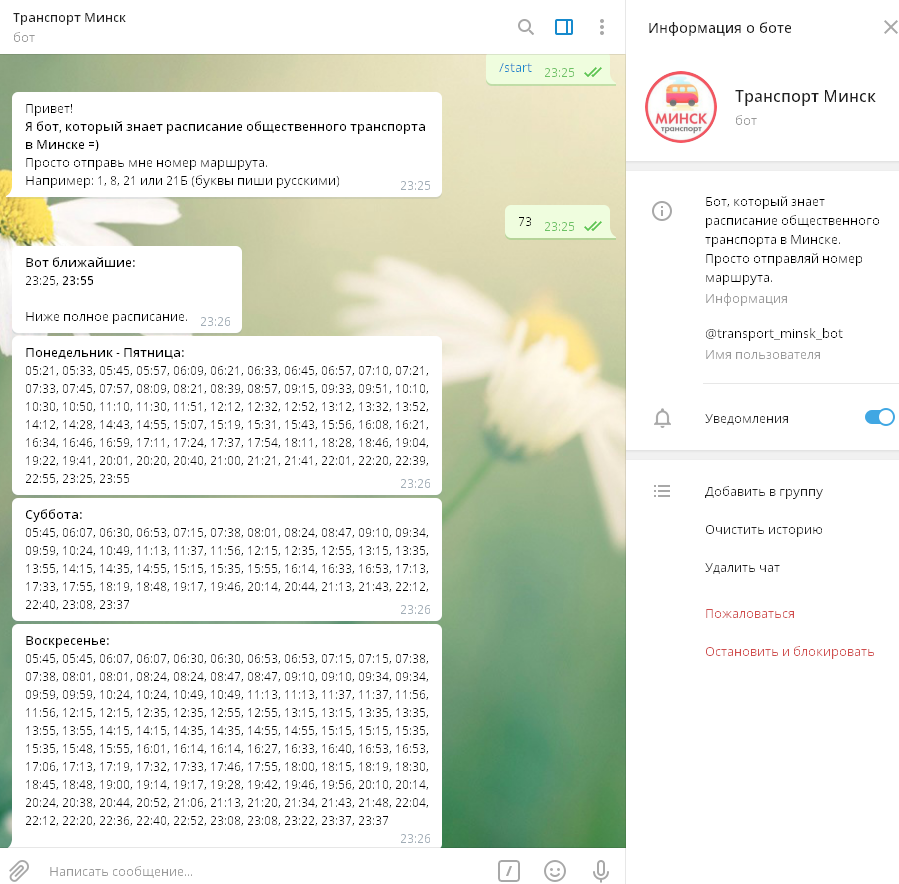
**

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.

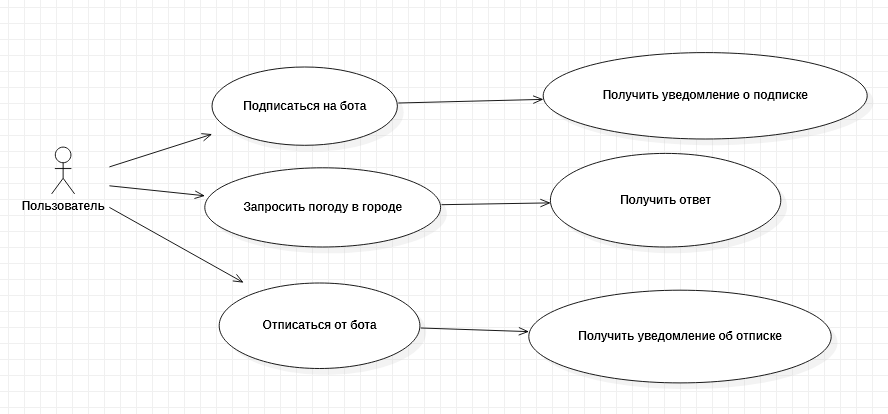


Рисунок 4.Диаграмма "Пользователь"

Пользователь заходит в чат с ботом, и у него есть 3 варианта.

При подписке на бота, он заносится в БД с активным статусом подписки.

При отписке, пользователь получает неактивный статус подписки, это было сделано для возможности в будущем введения платной подписки на рассылку.

При запросе погоды, пользователю предлагается ввести населённый пункт и ему выводит погоду в выбранном регионе

## 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.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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.

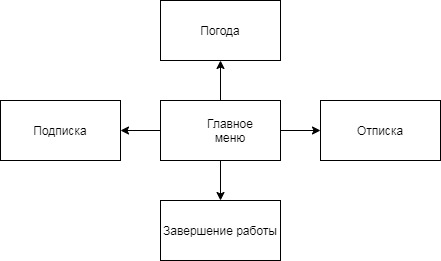


Рисунок 5. Сценарий использования

Пользователь после запуска программы может выполнить 3 действия: подписаться на 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.

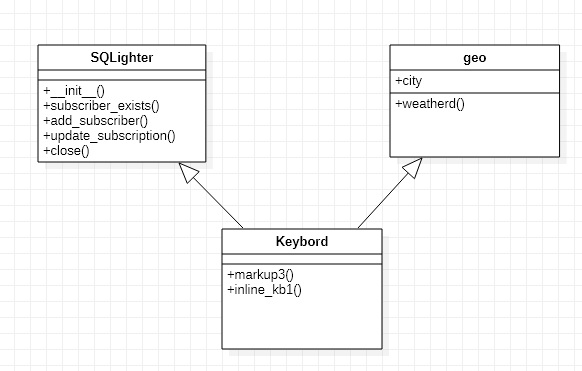


Рисунок 6. Диаграмма классов

Класс SQLiter содержит публичные методы get\_subscription(), subscriber\_exists(), add\_subscriber (), update\_subscribtion (), close () для работы с базой данных подписчиков.

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

Класс Keyboard содержит методы markup3 (), inline\_kb1 (), которые вызываются в главном модуле.

## 2.5. Описание главного модуля

В главный модуль входит реализация команд для бота.

В главный модуль импортируется 3 второстепенных модуля, один из них реализует работу с БД, один экранную клавиатуру, третий работу с библиотекой pyowm, то есть погодный API.

Также в главный модуль входит код, который запускает саму программу.

Листинг 1 Команда активации подписки

# Команда активации подписки

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

async def subscribe(message: types.Message):

"""проверяем есть ли подписка у юзера"""

if(not db.subscriber\_exists([message.from\_user.id](http://message.from_user.id/))):

"""оформляем подписку"""

db.add\_subscriber([message.from\_user.id](http://message.from_user.id/" \t "_blank))

else:

"""обновляем подписку юзера"""

db.update\_subscription([message.from\_user.id](http://message.from_user.id/" \t "_blank), True)

await message.answer("Вы подписались на бота")

Данная функция проверяет наличие подписки у пользователя и если ее нет, то оформляет ее.

Листинг 2 команда отписки

# Команда отписки

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

async def unsubscribe(message: types.Message):

"""проверяем есть ли отписан ли юзер"""

if(not db.subscriber\_exists([message.from\_user.id](http://message.from_user.id/" \t "_blank))):

"""выводим сообщение что он и так отписан"""

db.add\_subscriber([message.from\_user.id](http://message.from_user.id/" \t "_blank), False)

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

else:

"""отписываем юзера"""

db.update\_subscription([message.from\_user.id](http://message.from_user.id/" \t "_blank), False)

await message.answer("Вы отписаны от рассылки.") Данная функция проверяет наличие подписки у пользователя и если ее нет, то выводит сообщение о ее отсутствии, если же она имеется , то отписывает пользователя.

Блок-схема функции отписки.

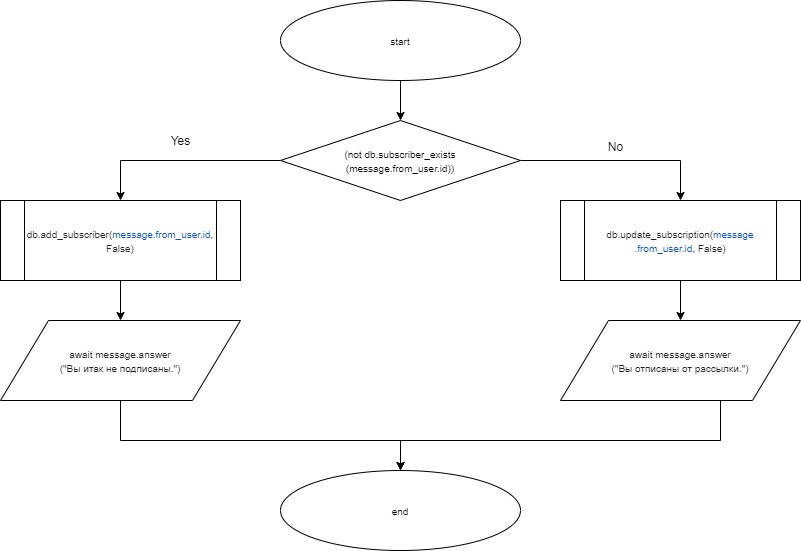


Рисунок 7. Блок-схема модуля отписки

Листинг 3 функция погоды

#погода

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

async def weather(message: types.Message):

"""получаем данные о городе"""

await message.answer("Введите город:")

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

async def get\_city(message):

"""сохраняем данные о городе"""

city = message.text

"""выводим погоду"""

await message.answer(geo.weatherd())

# запускаем лонг поллинг

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

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

Данная функция запрашивает у пользователя населенный пункт, и выводит погоду в нем.

Листинг 4 запуск бота

# запускаем лонг поллинг

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

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

## 2.6. Описание спецификаций к модулям

В данном разделе описаны спецификации к модулям. Спецификации к модулям заключается в следующем: использование методов библиотек aiogram, pyowm и sqlite3.

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.
* InlineKeyboardButton() – создание инлайн кнопки.
* KeyboardButton() – создание кнопки.
* sqlite3.connect() – подключение к БД.
* cursor() – создание объекта курсора.
* еxecute() – выполнение любых запросов sql.
* fetchall() – сохранение значений в переменной.
* сlose() – закрытие соединения с БД.

## 2.7. Описание модулей

Рассмотрим один из модулей программы - это модуль экранная клавиатура

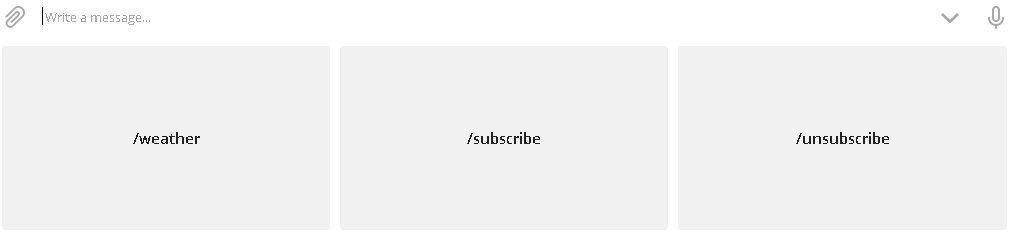


Рисунок 8. Экранная клавиатура

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

Листинг 5 модуля Keybord.py

from aiogram.types import InlineKeyboardButton, InlineKeyboardMarkup,\

ReplyKeyboardRemove, KeyboardButton, ReplyKeyboardMarkup

"""создаем инлайн кнопки (для кпестиков ноликов)"""

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')

"""создаем обычные кнопки"""

button10 = KeyboardButton('/weather')

button11 = KeyboardButton('/subscribe')

button12 = KeyboardButton('/unsubscribe')

"""собираем кнопки в 1 ряд"""

markup3 = ReplyKeyboardMarkup().row(

button10, button11, button12

)

"""создаем из кнопок поле 3 на 3"""

inline\_kb1 = InlineKeyboardMarkup()

inline\_kb1.add(

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

Листинг 6 модуля 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."

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

return answer

Листинг 7 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.

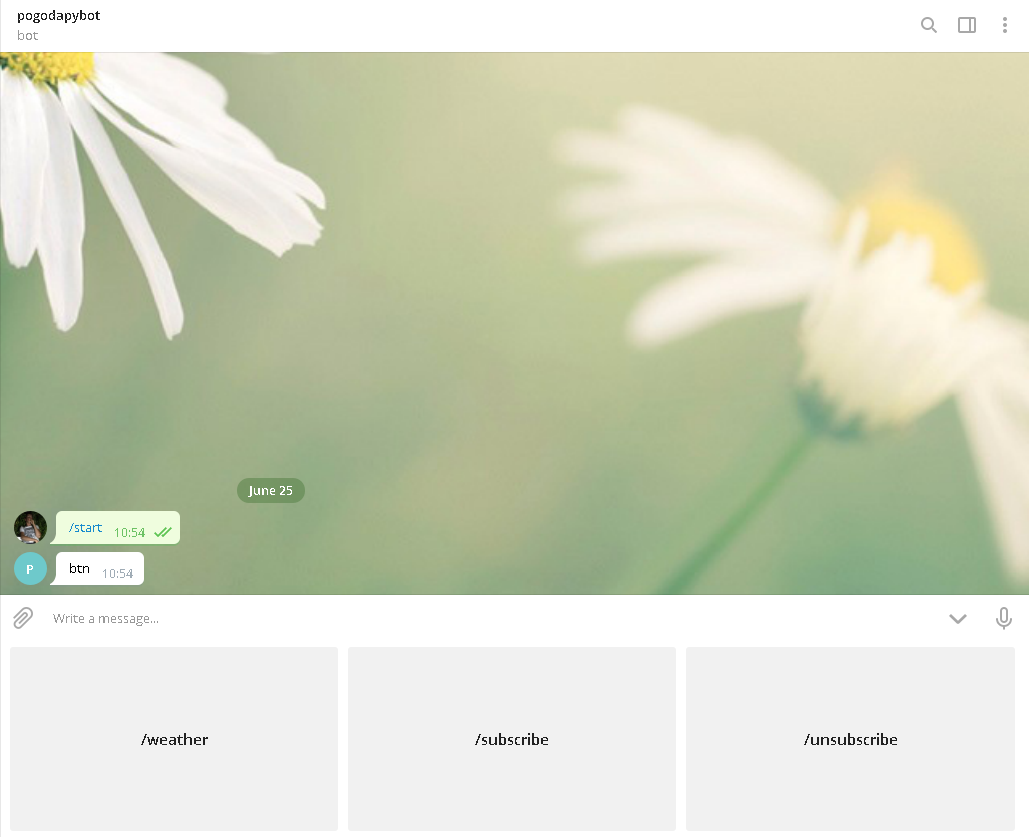
Шаг 1. На появившейся клавиатуре нажать кнопку /subscribe 

Рисунок 9. Экранная клавиатура с доступными командами

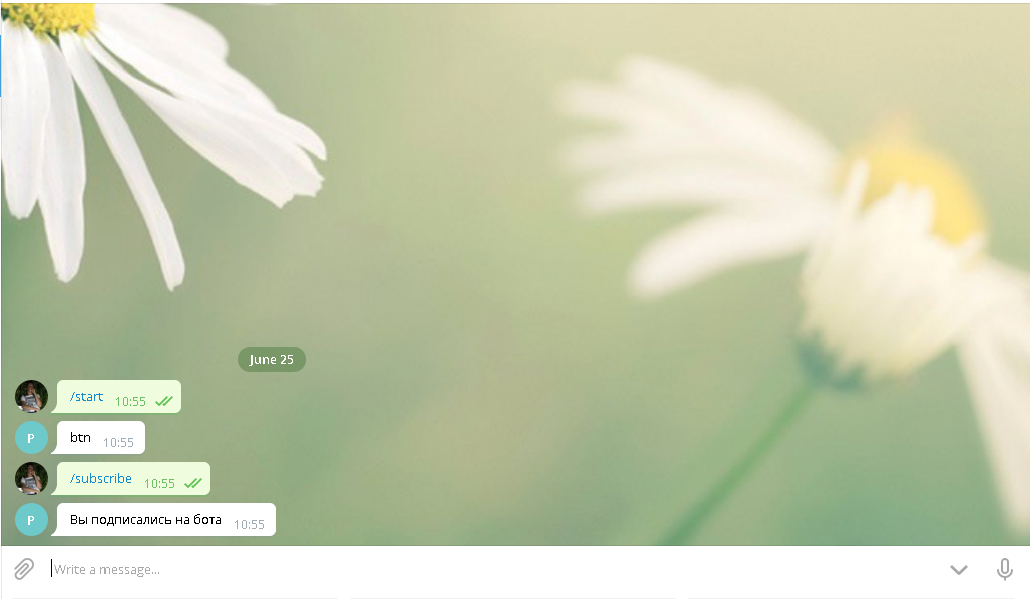


Рисунок 10.Результат работы /subscribe

Шаг 2. Нажать кнопку /weather и затем ввести город

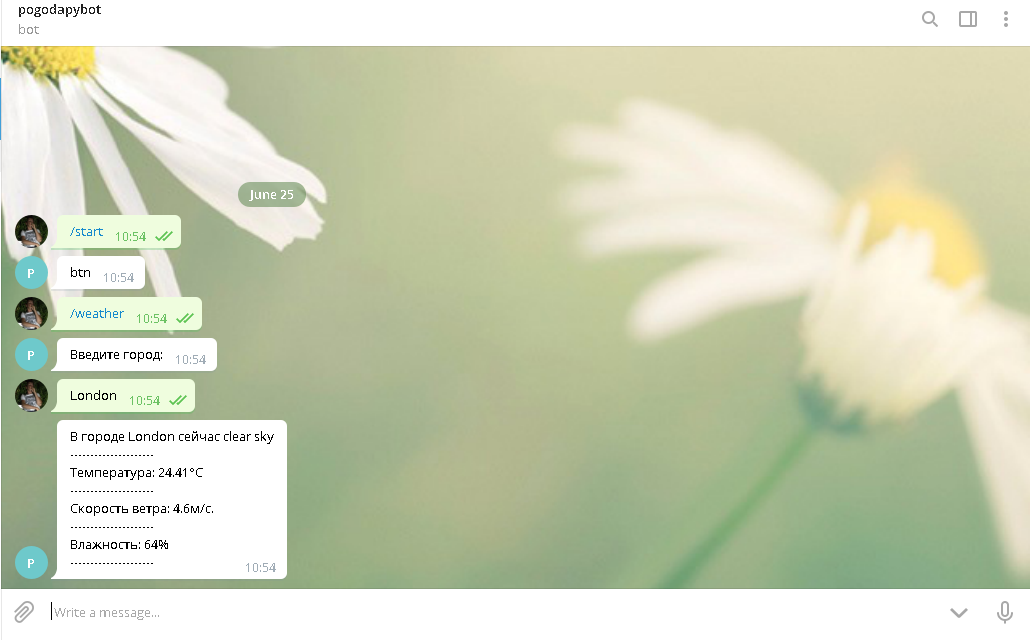


Рисунок 11. Результат работы /weather

Шаг 3. По желанию отписаться от бота (кнопка /unsubscribe)

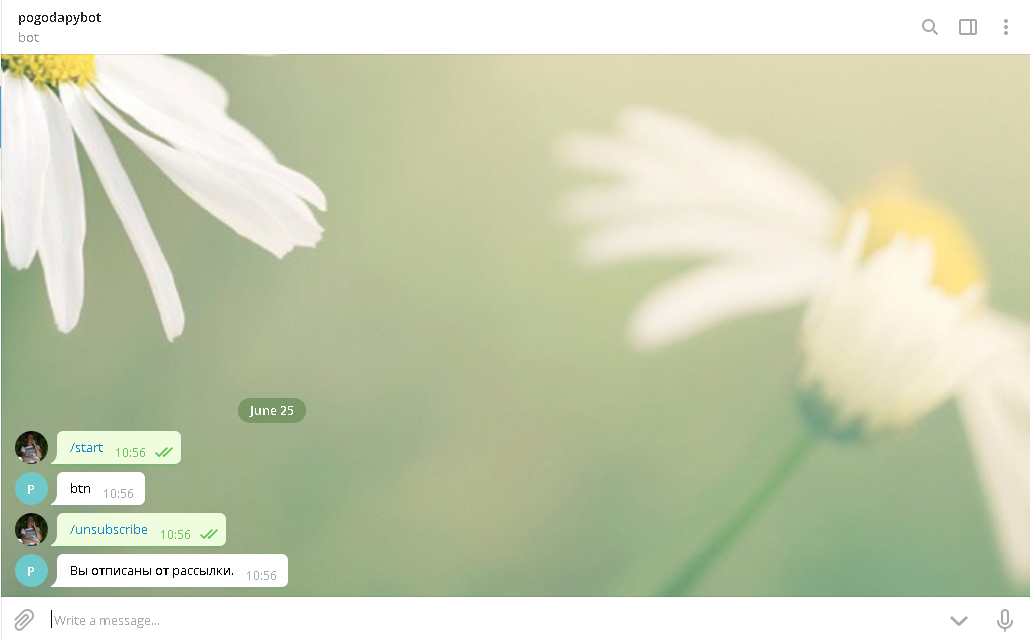


Рисунок 12. Результат работы /unsubscribe

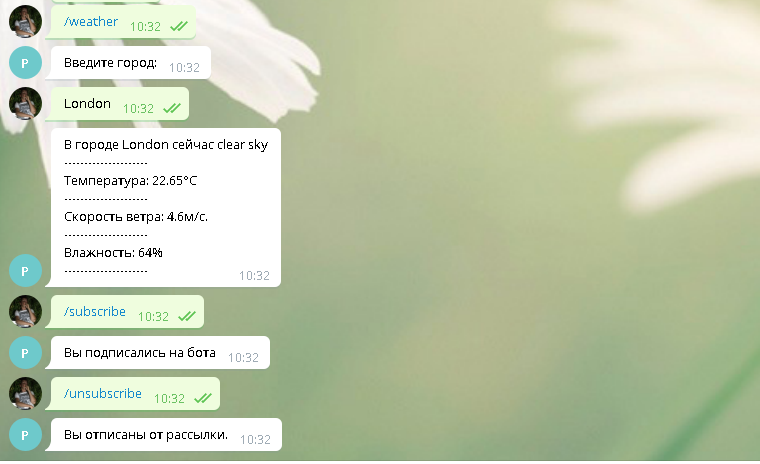


Рисунок 13. Пример работы бота

## 2.9. Описание применения средств отладки

При проведении тестов была выявлена ошибка, ее причиной стала устаревшая библиотека.

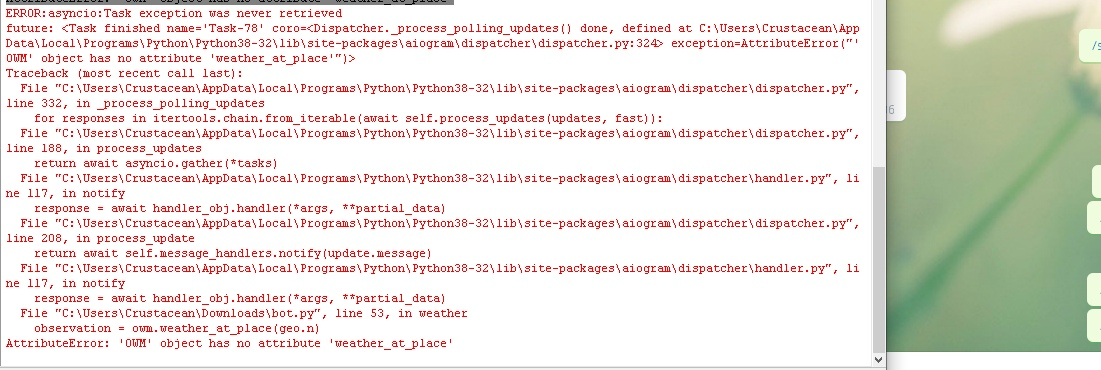


Рисунок 14. Ошибка при работе бота

После обновления ошибка была решена.

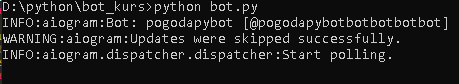


Рисунок 15. Решенная ошибка

## 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.

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 8 и Листинге 9

Листинг 8. Импорт объектов пример 1

#импорт модулей для работы с клавиатурой и кнопками

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

Листинг 9. Импорт объектов пример 2

from aiogram import Bot, Dispatcher, executor, types

#импорт модулей для работы с TelegramBotAPI

# Глава 3. Эксплуатационная часть

**Аннотация**

В данном программном документе приведено руководство оператора по применению и эксплуатации чат-бота Telegram, предназначенной для получения погоды.

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.Назначениепрограммы

* 1. Функциональное назначение программы

Чат-бот используется для показа погоды в введенном регионе.

* 1. Эксплуатационное назначение программы

Чат-бот может быть использован пользователями, которые часто пользуются мессенджером Telegram, для получения прогноза погоды.

1. Условия выполнения программы
   1. Минимальный состав аппаратных средств

ОС: Windows 10

Процессор: Как минимум 1 ГГц или SoC.

ОЗУ: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем).

Место на жестком диске: 16 ГБ (для 32-разрядных систем) или 20 ГБ (для 64-разрядных систем).

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

Дисплей: 800 x 600.

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

Требований нет.

1. Выполнение программы
   1. Загрузка и запуск программы

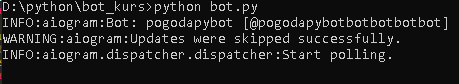
Перед использованием бота вам необходимо запустить сервер**

Рисунок 16. Запуск сервера

Далее вам нужно найти бота в поиске

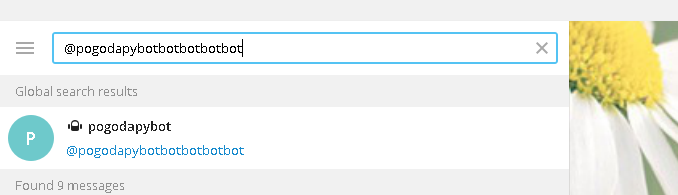


Рисунок 17. Поиск бота

* 1. Выполнение программы
     1. Выполнение функции подписка.

Нажмите на кнопку /subscribe, и вы подпишитесь на бота

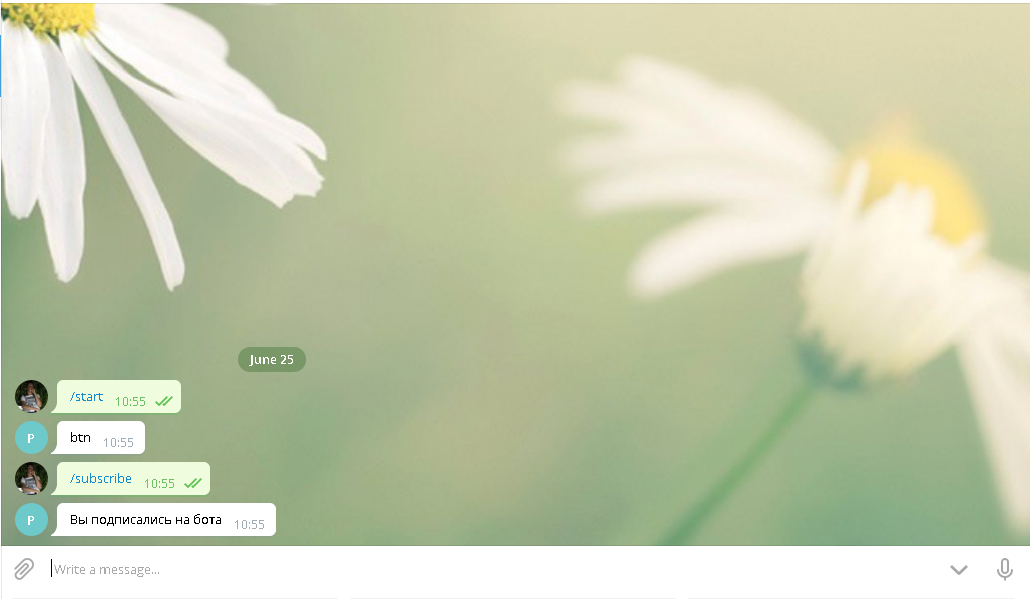


Рисунок 18. Подписка на бота

* + 1. Выполнение функции отписка.

Нажмите на кнопку /unsubscribe, и вы отпишетесь от бота

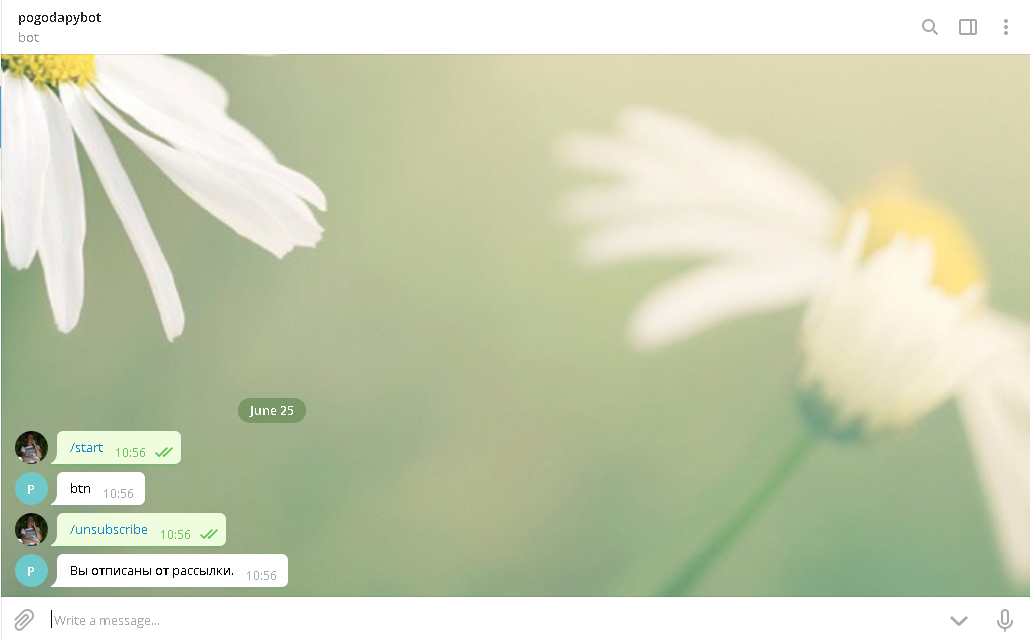


Рисунок 19. Отписка от бота

* + 1. Выполнение функции погода.

Нажмите на кнопку /weather, и вам предложат ввести населенный пункт для которого вы хотите запросить погоду, далее необходимо ввести населенный пункт, и вам выведет текущую погоду.

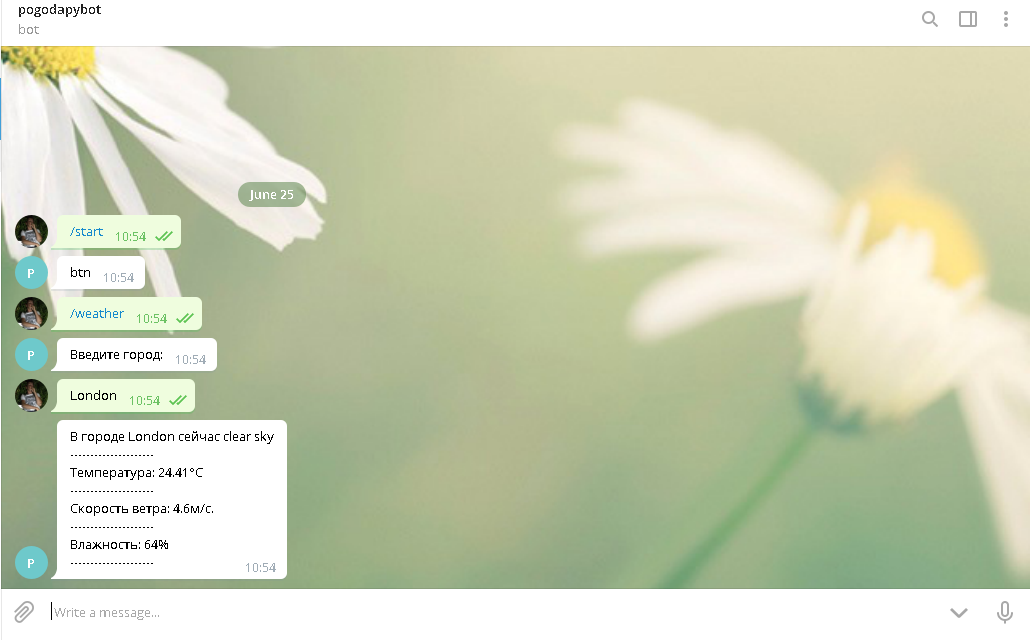


Рисунок 20. Погода

* 1. Завершение общения.

Чтобы завершить общение с ботом просто закройте диалог.

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 модуля.

import keybord as kb

import logging

import asyncio

import pyowm

import geo

import aiogram.types.base

from aiogram import Bot, Dispatcher, executor, types

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):

"""проверяем есть ли подписка у юзера"""

if(not db.subscriber\_exists([message.from\_user.id](http://message.from_user.id/" \t "_blank))):

"""оформляем подписку"""

db.add\_subscriber([message.from\_user.id](http://message.from_user.id/" \t "_blank))

else:

"""обновляем подписку юзера"""

db.update\_subscription([message.from\_user.id](http://message.from_user.id/" \t "_blank), True)

await message.answer("Вы подписались на бота")

# Команда отписки

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

async def unsubscribe(message: types.Message):

"""проверяем есть ли отписан ли юзер"""

if(not db.subscriber\_exists([message.from\_user.id](http://message.from_user.id/" \t "_blank))):

"""выводим сообщение что он и так отписан"""

db.add\_subscriber([message.from\_user.id](http://message.from_user.id/" \t "_blank), False)

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

else:

"""отписываем юзера"""

db.update\_subscription([message.from\_user.id](http://message.from_user.id/" \t "_blank), False)

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

#погода

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

async def weather(message: types.Message):

"""получаем данные о городе"""

await message.answer("Введите город:")

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

async def get\_city(message):

"""сохраняем данные о городе"""

city = message.text

"""выводим погоду"""

await message.answer(geo.weatherd())

#кнопки

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

async def buttons(message: types.Message):

"""выводим начальный набор кнопок"""  
await message.answer('btn', reply\_markup=kb.markup3)

#удаление кнопок

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

async def removekb(message: types.Message):

"""удаляем начальный набор кнопок"""

await message.answer('remove\_kb', reply\_markup=kb.ReplyKeyboardRemove())

# запускаем лонг поллинг

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

"""создаем инлайн кнопки (для кпестиков ноликов)"""

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')

"""создаем обычные кнопки"""

button10 = KeyboardButton('/weather')

button11 = KeyboardButton('/subscribe')

button12 = KeyboardButton('/unsubscribe')

"""собираем кнопки в 1 ряд"""

markup3 = ReplyKeyboardMarkup().row(

button10, button11, button12

)

"""создаем из кнопок поле 3 на 3"""

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

"""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

# 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()

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)