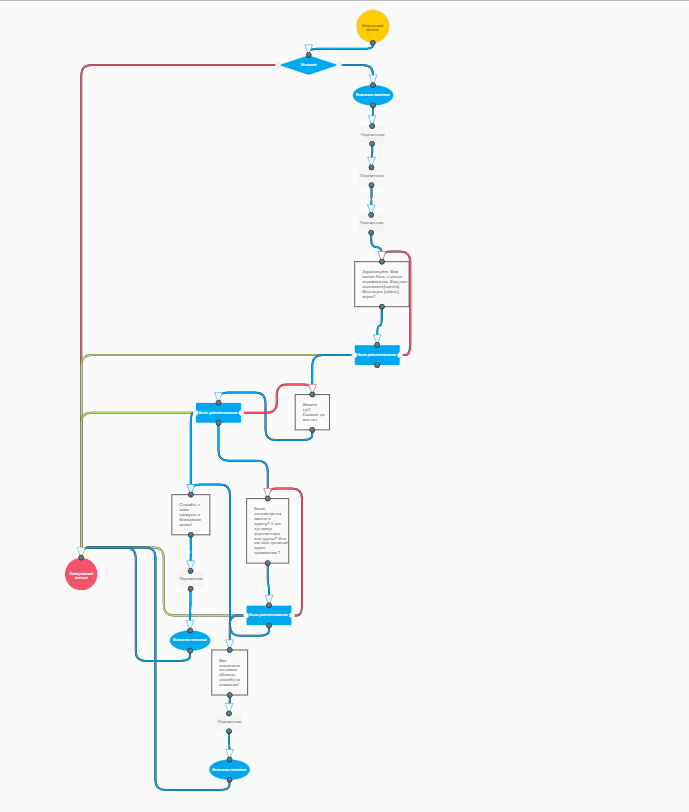
**1.**



Данная схема создана для сценария обзвон-должников.

Задача сценария: верификация абонента (должник/не должник)

Красные ветки – означают переход по умолчанию и отслеживают тишину от абонента.

Зеленые ветки – отвечают за негативную реакцию абонента.

Блок исходящий звонок(оранжевый) – запуск голосового робота, который звонит абоненту

Блок завершение вызова(красный) – заканчивает работу голосового помощника и сбрасывает абонента

Условие(ромб) – отвечает за то, взял ли абоненте трубку. Если нет – по умолчанию сбрасываем. Если да, то идем по сценарию.

Внешняя система(овал) – переход на внутренние микро сервисы компании(база данных, ексель файл, json-файл и т.д.) из которых можно взять информацию, а также, занести её.

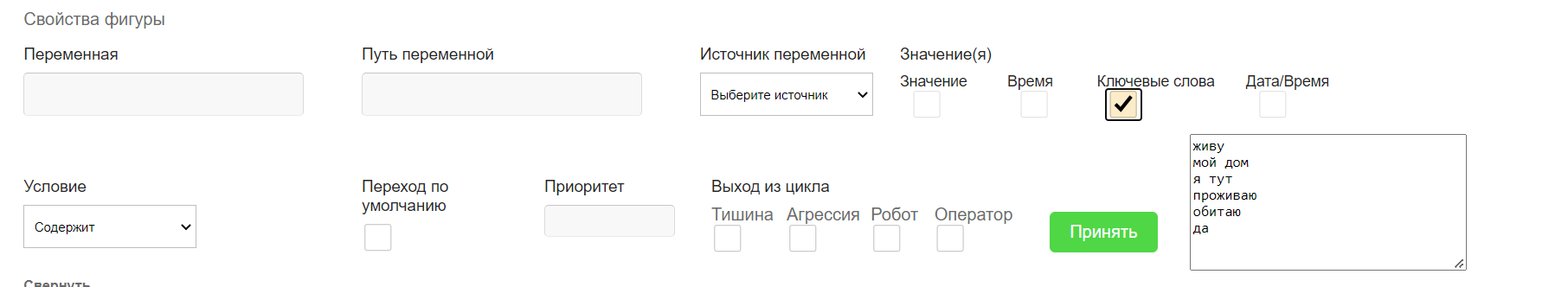
Блок распознавания(прямоугольник) – отвечает за переход к микро сервису, который отвечает за обработку слов от абонента, и возвращает значение в понятном для человека тексте(обычный текст)

Ответ абоненту(белые блоки с текстом) – в них заложен текст, который по определенному сценарию будет сказан абоненту.

Блоки переменных(маленькие серые блоки) – в них хранятся переменные, которые мы можем передать во внешнюю систему или из внешней системы сохранить данные в них. Их функциональность ничем не отличается от функциональности любого продвинутого языка программирования.

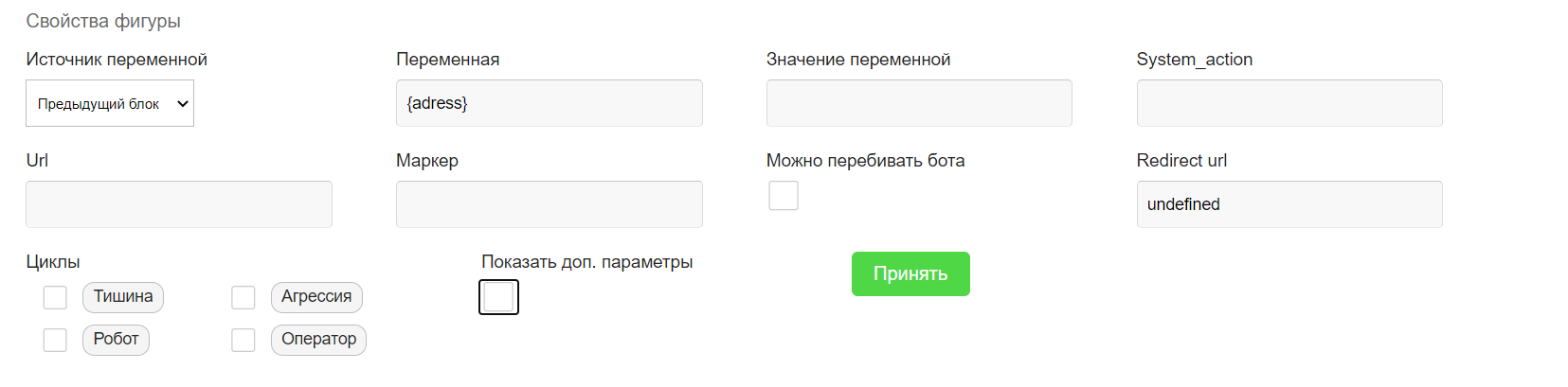
Простыми словами, как все работает:

В ветках хранятся значения и адрес к микро сервису(предположим, что это некий json файл). В ветках стоят свои условия, которые проверяют на содержание фразы:

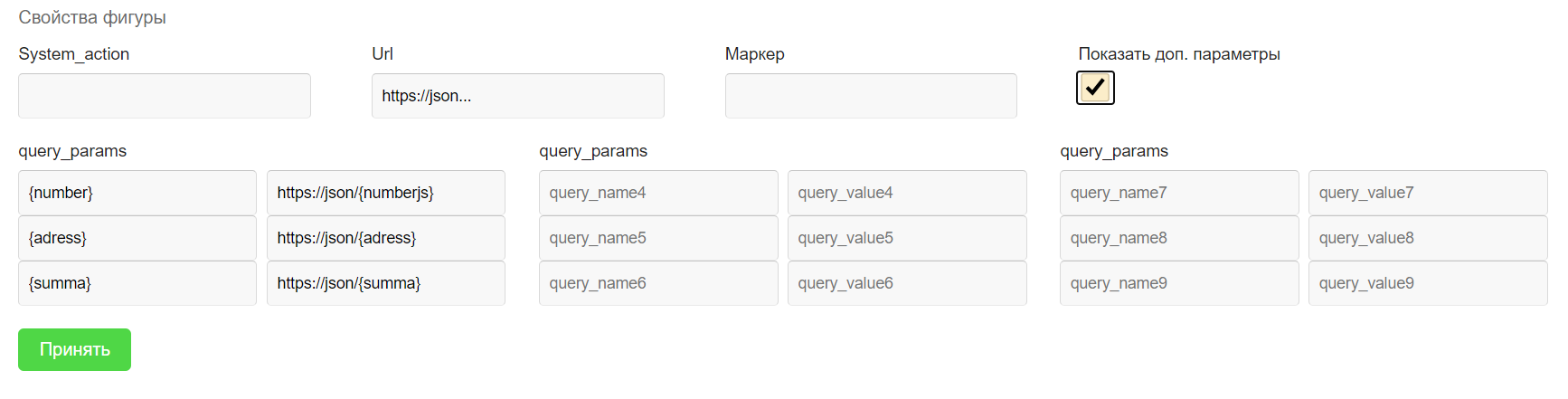


С помощью веток мы переходим по сценарию.

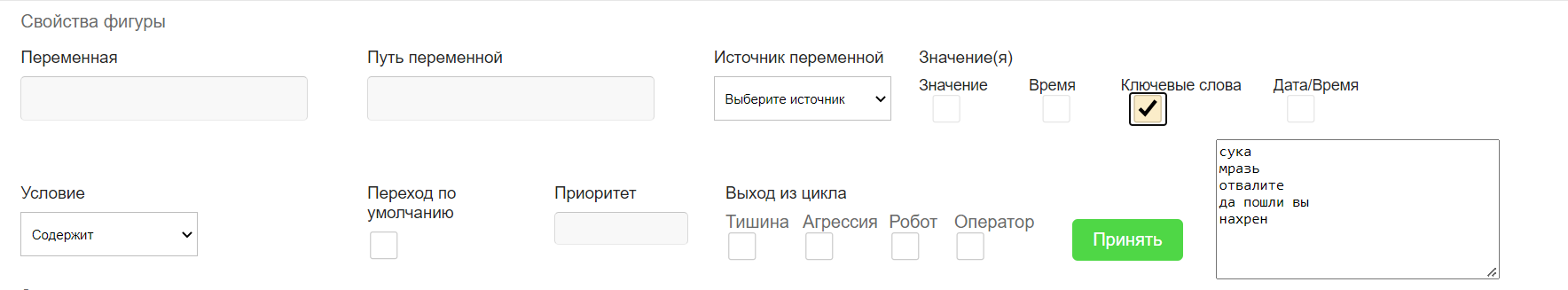
В переменных также используются внутренние свойства:



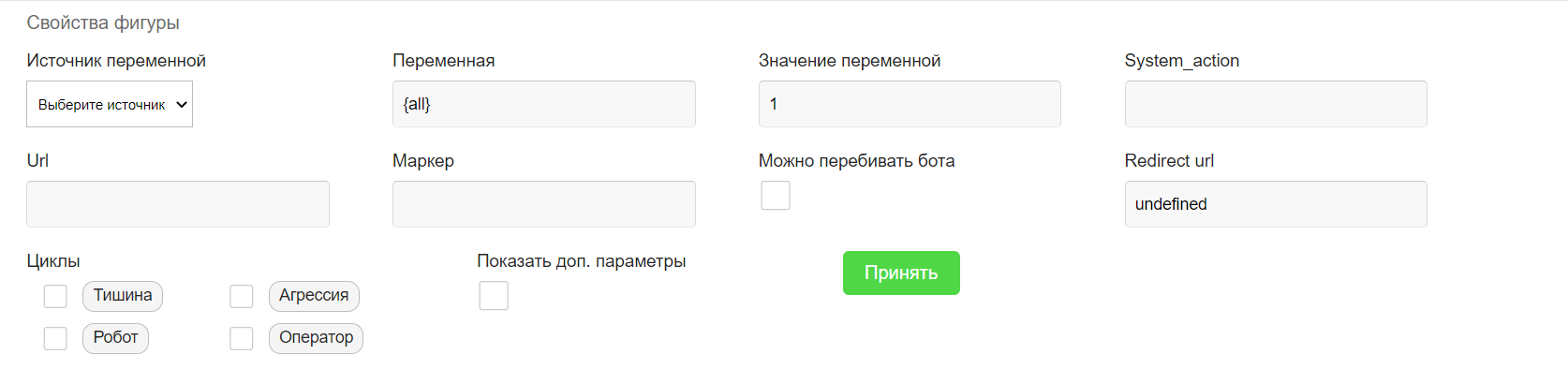
Во блоке внешней системы также используются свойства подключения и взаимодействия:

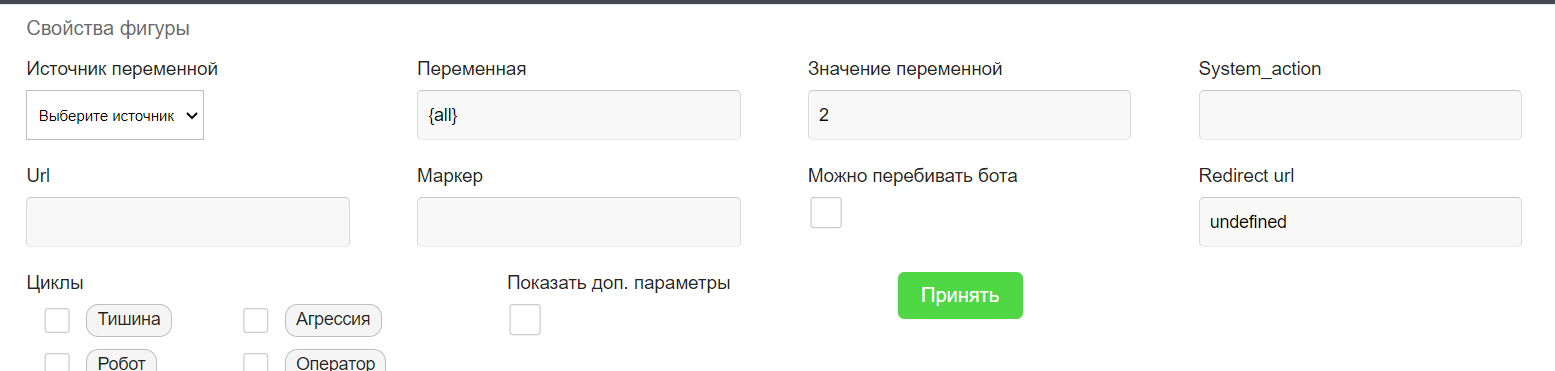


Зеленая ветка, пример свойств – содержание ключевых слов:



Перед тем, как бот завершит вызов, мы передаем значения в микро сервис через блок внешней системы. Переменная {all} = 1, если должник, {all} = 2 если не должник:





Предполагаем, что взаимодействие через блоки внешней системы – происходят на бэкэнде в виде php запросов.

**2. ТГ-БОТ**

Код программы:

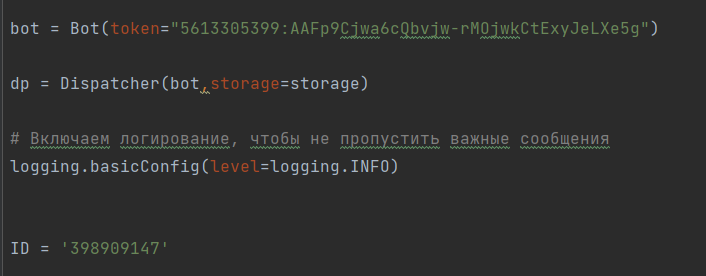
import logging  
from datetime import datetime  
import time  
import sqlite3 as sq  
import sqlite3  
import asyncio  
import os  
import threading  
from datetime import date  
from aiogram import Bot, Dispatcher, executor, types  
from aiogram.dispatcher.filters import Text  
import requests  
from aiogram.contrib.fsm\_storage.memory import MemoryStorage  
from aiogram.dispatcher import FSMContext  
from time import sleep  
from aiogram.dispatcher.filters.state import State, StatesGroup  
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton, ReplyKeyboardMarkup, KeyboardButton  
  
storage = MemoryStorage()  
  
  
bot = Bot(token="5613305399:AAFp9Cjwa6cQbvjw-rMOjwkCtExyJeLXe5g")  
  
dp = Dispatcher(bot,storage=storage)  
#dp2= Dispatcher(bot,storage=storage)  
  
# Включаем логирование, чтобы не пропустить важные сообщения  
logging.basicConfig(level=logging.INFO)  
  
  
ID = '398909147'  
  
  
  
  
@dp.message\_handler(commands='add')  
async def add\_func(message: types.Message):  
 global base, cur  
 base = sq.connect('main.db')  
 cur = base.cursor()  
 if base:  
 print('Data base connected OK!')  
 base.execute("CREATE TABLE IF NOT EXISTS info(id INTEGER, datatime REAL, name TEXT, phone INTEGER,PRIMARY KEY(id AUTOINCREMENT))")  
  
 string1 = message.text  
 string2 = string1[:0]+string1[4:]  
 string2 = string2.strip()  
 strings = string2.split(';')  
 dat = datetime.strptime(strings[0],"%d.%m.%Y") #%H:%M:%S  
 # timestamp = strings[0]  
 # value = datetime.fromtimestamp(timestamp)  
 #print(value.strftime('%Y-%m-%d %H:%M:%S'))  
 sqlite\_insert = """INSERT INTO info(datatime,name,phone) VALUES(?,?,?)"""  
 data\_tuple = (dat, strings[1], strings[2])  
 cur.execute(sqlite\_insert, data\_tuple)  
 base.commit()  
 [var\_id,] = cur.execute('SELECT max(id) FROM info').fetchone()  
 base.commit()  
 if (strings[2]):  
 await bot.send\_message(message.chat.id,f"Строка с id = {var\_id} создана")  
  
  
  
  
@dp.message\_handler(commands='del')  
async def del\_func(message: types.Message):  
 string1 = message.text  
 string2 = string1[:0] + string1[4:]  
 string2 = string2.strip()  
 sqlite\_delete = """DELETE FROM info WHERE id = ?"""  
 data\_tuple = (string2,)  
 cur.execute(sqlite\_delete, data\_tuple)  
 base.commit()  
 await bot.send\_message(message.chat.id, f"Строка с id = {string2} удалена")  
  
  
  
@dp.message\_handler(commands='sel')  
async def sel\_func(message: types.Message):  
 string1 = message.text  
 string2 = string1[:0] + string1[4:]  
 string2 = string2.strip()  
 [sqlite\_sel,] = cur.execute("SELECT datatime,name,phone FROM info WHERE id = ?",(string2,))  
 base.commit()  
 await bot.send\_message(message.chat.id, f"Строка с id = {string2}\ndatatime = {sqlite\_sel[0]}\nname = {sqlite\_sel[1]}\nphone = {sqlite\_sel[2]}")  
  
  
  
@dp.message\_handler(commands='help')  
async def help\_func(message: types.Message):  
 await bot.send\_message(ID,  
 f"Для того, чтобы добавить новую запись в бд используйте команду /add - далее ставите пробел и перечисляете дату,имя и номер телефона через <<;>> Пример - /add 01-01-2022;Иван;8918000000\n"  
 f"Для того, чтобы удалить запись из бд используйте команду /del - ставите пробел и пишите id. Пример - /del 5\n"  
 f"Для того, чтобы посмотреть нужную строчку из бд - используйте команду /sel - ставите пробел и пишите id. Пример - /sel 5")  
  
  
  
async def main():  
 #executor.start\_polling(dispatcher=dp2)  
 print('Бот вышел в онлайн')  
 global base1, cur1  
 base1 = sq.connect('main.db')  
 cur1 = base1.cursor()  
 time = 0.1  
 # if base:  
 # print('Data base connected OK!')  
 # base.execute(  
 # "CREATE TABLE IF NOT EXISTS info(id INTEGER, datatime REAL, name TEXT, phone INTEGER,PRIMARY KEY(id AUTOINCREMENT))")  
 [var\_last, ] = cur1.execute('SELECT max(id) FROM info').fetchone()  
 print(f"var\_last = {var\_last}")  
 base1.commit()  
 looptime = asyncio.get\_running\_loop()  
 end\_time = loop.time() + 5.0  
 while True:  
  
 base1 = sq.connect('main.db')  
 cur1 = base1.cursor()  
 [var\_max, ] = cur1.execute('SELECT max(id) FROM info').fetchone()  
 base1.commit()  
 print(f"var\_max = {var\_max}")  
  
 if var\_last != var\_max:  
 [var\_new, ] = cur1.execute("SELECT datatime,name,phone FROM info WHERE id = ?", (var\_max,))  
 base1.commit()  
 print("True")  
 await bot.send\_message(ID,  
 f"Строка с id = {var\_max}\ndatatime = {var\_new[0]}\nname = {var\_new[1]}\nphone = {var\_new[2]}")  
 var\_last = var\_max  
 else:  
 print("False")  
  
 await asyncio.sleep(5)  
 print("SLEEEEP")  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 loop = asyncio.get\_event\_loop()  
 loop.create\_task(main())  
 executor.start\_polling(dp)

Как работает:

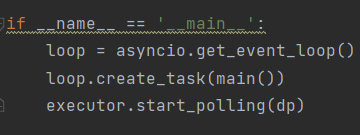
Подключаем асинхронную библиотеку aiogram, для того, чтобы ускорить работу бота, а также сделать его функционал не последовательным. Библиотека datetime – нужна нам, чтобы конвертировать через unix timestamp.

Подвязываем бота через <https://t.me/BotFather> в тг - это главный бот. Получаем токен и используем его в нашем коде.

Идентификация и подключение бота:



Запуск бота, чтобы работал постоянно:



Главная функция бота, которая срабатывает при запуске сразу:

async def main():  
 #executor.start\_polling(dispatcher=dp2)  
 print('Бот вышел в онлайн')  
 global base1, cur1  
 base1 = sq.connect('main.db')  
 cur1 = base1.cursor()  
 time = 0.1  
 # if base:  
 # print('Data base connected OK!')  
 # base.execute(  
 # "CREATE TABLE IF NOT EXISTS info(id INTEGER, datatime REAL, name TEXT, phone INTEGER,PRIMARY KEY(id AUTOINCREMENT))")  
 [var\_last, ] = cur1.execute('SELECT max(id) FROM info').fetchone()  
 print(f"var\_last = {var\_last}")  
 base1.commit()  
 looptime = asyncio.get\_running\_loop()  
 end\_time = loop.time() + 5.0  
 while True:  
  
 base1 = sq.connect('main.db')  
 cur1 = base1.cursor()  
 [var\_max, ] = cur1.execute('SELECT max(id) FROM info').fetchone()  
 base1.commit()  
 print(f"var\_max = {var\_max}")  
  
 if var\_last != var\_max:  
 [var\_new, ] = cur1.execute("SELECT datatime,name,phone FROM info WHERE id = ?", (var\_max,))  
 base1.commit()  
 print("True")  
 await bot.send\_message(ID,  
 f"Строка с id = {var\_max}\ndatatime = {var\_new[0]}\nname = {var\_new[1]}\nphone = {var\_new[2]}")  
 var\_last = var\_max  
 else:  
 print("False")  
  
 await asyncio.sleep(5)  
 print("SLEEEEP")

В ней мы подключаемся к нашей бд, настраиваем курсом и проверяем на успех подключение.

Также создаем таблицу info. В этой главной функции прописан код, которые в течении каждых 5 секунд проверяет, есть ли новая запись в бд, которая была введена вручную. Сделано это через бесконечный цикл While и SQL-запросы.

Команда add:

@dp.message\_handler(commands='add')  
async def add\_func(message: types.Message):  
 string1 = message.text  
 string2 = string1[:0]+string1[4:]  
 string2 = string2.strip()  
 strings = string2.split(';')  
 dat = datetime.strptime(strings[0],"%d.%m.%Y") #%H:%M:%S  
 # timestamp = strings[0]  
 # value = datetime.fromtimestamp(timestamp)  
 #print(value.strftime('%Y-%m-%d %H:%M:%S'))  
 sqlite\_insert = """INSERT INTO info(datatime,name,phone) VALUES(?,?,?)"""  
 data\_tuple = (dat, strings[1], strings[2])  
 cur.execute(sqlite\_insert, data\_tuple)  
 base.commit()  
 [var\_id,] = cur.execute('SELECT max(id) FROM info').fetchone()  
 base.commit()  
 if (strings[2]):  
 await bot.send\_message(message.chat.id,f"Строка с id = {var\_id} создана")

Команда del:

@dp.message\_handler(commands='del')  
async def del\_func(message: types.Message):  
 string1 = message.text  
 string2 = string1[:0] + string1[4:]  
 string2 = string2.strip()  
 sqlite\_delete = """DELETE FROM info WHERE id = ?"""  
 data\_tuple = (string2,)  
 cur.execute(sqlite\_delete, data\_tuple)  
 base.commit()  
 await bot.send\_message(message.chat.id, f"Строка с id = {string2} удалена")

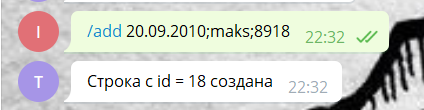
Команда sel:

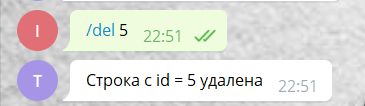
@dp.message\_handler(commands='sel')  
async def sel\_func(message: types.Message):  
 string1 = message.text  
 string2 = string1[:0] + string1[4:]  
 string2 = string2.strip()  
 [sqlite\_sel,] = cur.execute("SELECT datatime,name,phone FROM info WHERE id = ?",(string2,))  
 base.commit()  
 await bot.send\_message(message.chat.id, f"Строка с id = {string2}\ndatatime = {sqlite\_sel[0]}\nname = {sqlite\_sel[1]}\nphone = {sqlite\_sel[2]}")

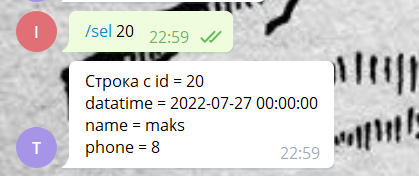
Команда help:

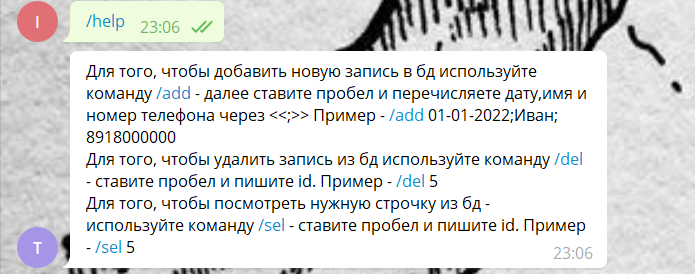
@dp.message\_handler(commands='help')  
async def help\_func(message: types.Message):  
 await bot.send\_message(message.chat.id,  
 f"Для того, чтобы добавить новую запись в бд используйте команду /add - далее ставите пробел и перечисляете дату,имя и номер телефона через <<;>> Пример - /add 01-01-2022;Иван;8918000000\n"  
 f"Для того, чтобы удалить запись из бд используйте команду /del - ставите пробел и пишите id. Пример - /del 5\n"  
 f"Для того, чтобы посмотреть нужную строчку из бд - используйте команду /sel - ставите пробел и пишите id. Пример - /sel 5")

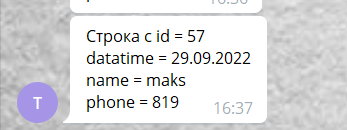
Пример работы бота:











Скрины бд:

