

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ
по лабораторной работе №4
по дисциплине «ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ KOTLIN»
ТЕМА: Подключение базы данных к Telegram-боту.

Студент гр. 0302

Кузнецов К.Е.

Студент гр. 0302

Головатюк К.А.

Преподаватель

Кулагин М.В.

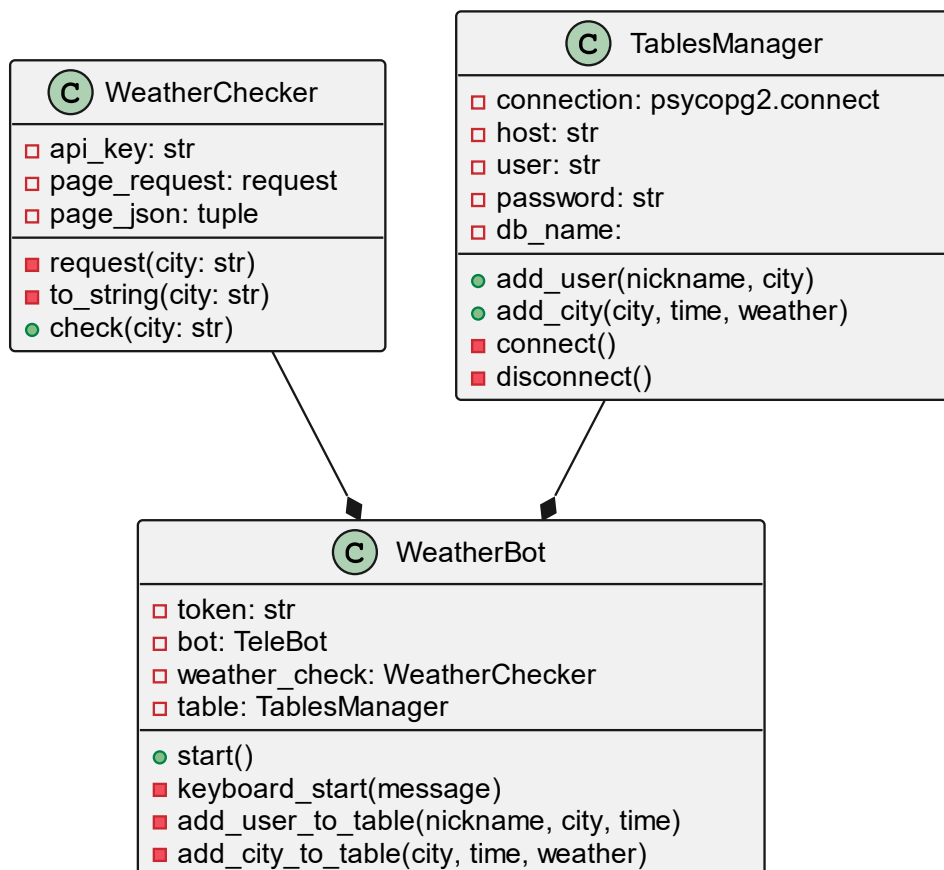
Санкт-Петербург

2022

Цель работы

В рамках практической работы требуется реализовать взаимодействие разработанного бота с таблицами в базе данных.

Спецификация программы



Описание интерфейса пользователя программы

Пользователю выводится информация и кнопка “Погода”. После нажатия на кнопку, требуется ввести название города. Если название города введено верно, то выводится погода в этом городе, иначе выводится информация об ошибке. Если запрос успешен, в базу данных (PostgreSQL) заносится никнейм пользователя, город, время запроса и погода. Так же была написана программа инициализации таблицы.

Текст программы

main.py:

```
from controler import WeatherBot

if __name__ == '__main__':
    test = WeatherBot()
    test.start()
    pass
```

controler.py:

```
import telebot
from weather import WeatherChecker
from telebot import types
from tablesManager import TablesManager
from datetime import datetime

class WeatherBot:
    def __init__(self, path_weather_api_key="weather_api_key",
path="telegram_bot_token"):
        self.__token = None
        try:
            self.__token = open(path, "r").read()
        except FileNotFoundError:
            self.__token = None
            print("Неверный путь")
            return
        self.__bot = telebot.TeleBot(self.__token)
        self.__weather_check =
WeatherChecker(path_weather_api_key)
        self.__table = TablesManager()

        @self.__bot.message_handler(content_types=['text'])
        def start(message):
            self.__keyboard_start(message)
```

```

def weather(message):
    result = self.__weather_check.check(message.text)
    self.__bot.send_message(message.from_user.id,
result[1])

    if result[0]:
        self.__add_city_to_table(message.text,
datetime.now(), result[1])

self.__add_user_to_table(message.from_user.username,
message.text, datetime.now())
    self.__keyboard_start(message)

    @self.__bot.callback_query_handler(func=lambda call:
True)
    def callback_worker(call):
        if call.data == "weather":
            self.__bot.send_message(call.message.chat.id,
'Введите название города.')

self.__bot.register_next_step_handler(call.message, weather)

def __keyboard_start(self, message):
    keyboard = types.InlineKeyboardMarkup()
    key_weather = types.InlineKeyboardButton(text='Погода',
callback_data='weather')
    keyboard.add(key_weather)
    text = 'Нажми на кнопку'
    self.__bot.send_message(message.from_user.id, text=text,
reply_markup=keyboard)

def __add_user_to_table(self, nickname, city, time):
    self.__table.add_user(nickname, city, time)

def __add_city_to_table(self, city, time, weather):

```

```

self.__table.add_city(city, time, weather)

def start(self):
    if self.__token is None:
        print("Нет токена")
    self.__bot.polling(none_stop=True, interval=0)

```

weather.py:

```

import requests
import json

# icons = ["☀️", "🌤️", "🌥️", "☁️", "⛅️", "🌧️", "🌩️", "⚡️", "❄️",
"🌫️"]

```

```

icons = {"01d": "☀️", "01n": "☀️",
         "02d": "🌤️", "02n": "🌤️",
         "03d": "☁️", "03n": "☁️",
         "04d": "☁️", "04n": "☁️",
         "09d": "🌧️", "09n": "🌧️",
         "10d": "🌩️", "10n": "🌩️",
         "11d": "⚡️", "11n": "⚡️",
         "13d": "❄️", "13n": "❄️",
         "50d": "🌫️", "50n": "🌫️"}

```

```

class WeatherChecker:
    def __init__(self, path="weather_api_key"):
        self.__api_key = None
        self.__page_request = None
        self.__page_json = None
        try:
            self.__api_key = open(path, "r").read()
        except FileNotFoundError:
            print("Неверный путь")

```

```

def __request(self, city):
    self.__page_request = requests.get(
        f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={self.__api_key}")
    if self.__page_request.status_code != 200:
        return False

    self.__page_json = json.loads(self.__page_request.text)
    return True

def __to_string(self, city):
    return str(f"{city}:
{icons[self.__page_json['weather'][0]['icon']]} \n"
f"Температура:
{int(self.__page_json['main']['temp'] - 273.15)} °C, \n"
f"ощущается как:
{int(self.__page_json['main']['feels_like'] - 273.15)} °C. \n"
f"Влажность:
{self.__page_json['main']['humidity']}% \n"
f"Скорость ветра:
{self.__page_json['wind']['speed']} км/ч. \n"
)

def check(self, city):
    if self.__api_key is None:
        return "Нет ключа api!!!"
    if self.__request(city) is False:
        return "Ошибка запроса!!!"
    return self.__to_string(city)

```

tablesManager.py

```
import psycpg2
```

```

class TablesManager:

    def __init__(self):

        self.__connection = None

        self.__host = "127.0.0.1"

        self.__user = "postgres"

        self.__password = "123456789"

        self.__db_name = "postgres"


    def add_user(self, nickname, city):

        self.__connect()

        if not self.__connection.closed:

            try:

                with self.__connection.cursor() as cursor:

                    cursor.execute("""SELECT id FROM cities
WHERE (city='%s')""" % (city))

                    id = 0

                    for i in cursor:

                        id = i[0]

                    cursor.execute("""INSERT INTO users
(nickname, city_id) VALUES ('%s', %d)""" %
                                (nickname, id))

                self.__connection.commit()

            except Exception as ex:

```

```

        print("While adding user exception: ", ex)

    self.__disconnect()

def add_city(self, city, time, weather):
    self.__connect()

    if not self.__connection.closed:
        try:
            with self.__connection.cursor() as cursor:
                cursor.execute(
                    """INSERT INTO cities (city,
time_searching, weather) VALUES
                    ('%s', '%s', '%s')""" %
                    (city, time, weather)
                )

            self.__connection.commit()

        except Exception as ex:
            print("While adding city exception: ", ex)

    self.__disconnect()

def __connect(self):
    try:
        self.__connection = psycopg2.connect(
            host=self.__host,
            user=self.__user,

```



```

        password=self.__password,

        database=self.__db_name

    )

except Exception as ex:

    self.__connection.close()

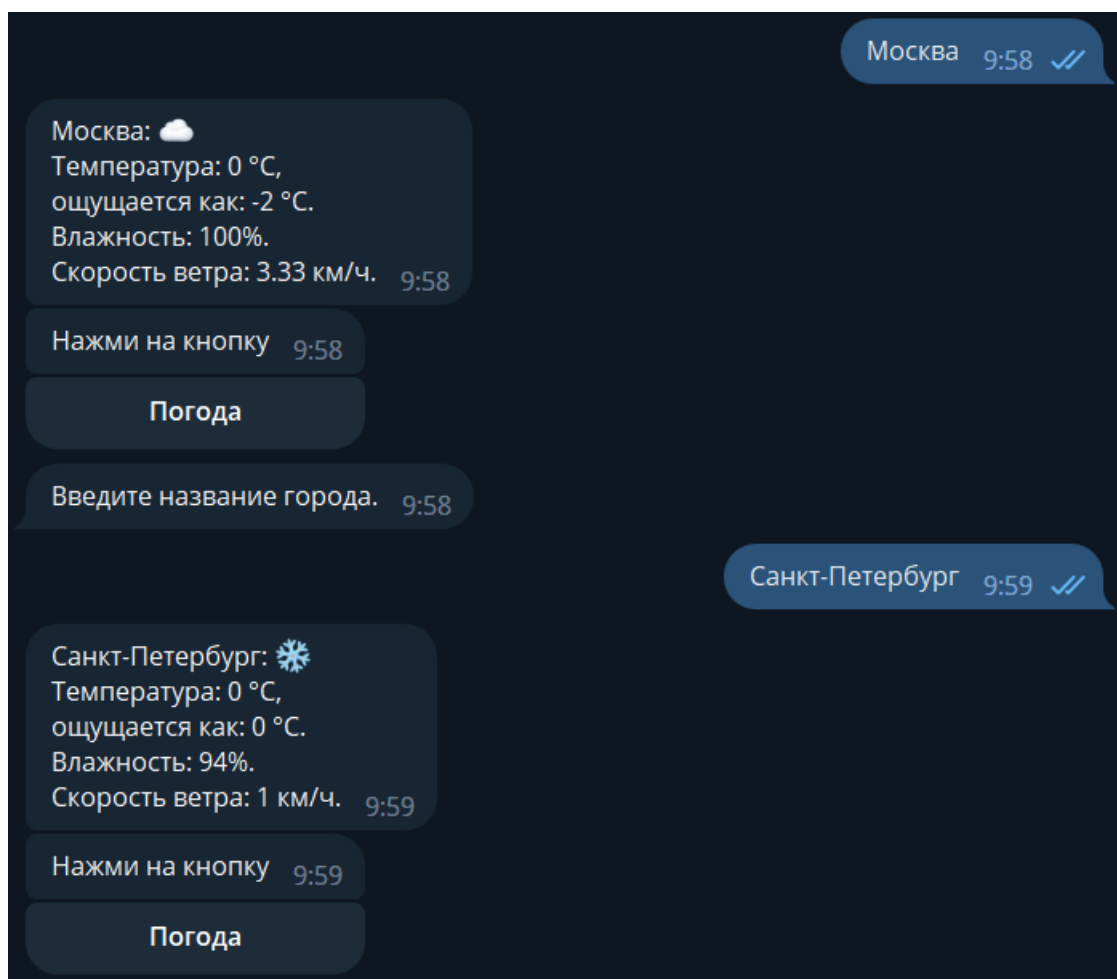
    print("Table connection failed. Exception: ", ex)

def __disconnect(self):

    self.__connection.close()

```

Пример работы



Выводы

В ходе данной работы был получен навык интеграции базы данных в программу.