МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ХПІ»

Кафедра «Обчислювальна техніка та програмування»

Розрахунково-графічне завдання. Написання телеграм бота на Python

Виконав:

Студент групи КІТ-118г

Кущ О.Е.

Перевірив:

Давидов В.В.

Харків 2021

Зміст

[Вступ 3](#_Toc72762490)

[Необхідність телеграм боту. 4](#_Toc72762491)

[Завдання 4](#_Toc72762492)

[Практична частина 4](#_Toc72762493)

[Команда add 7](#_Toc72762494)

[Команда list 9](#_Toc72762495)

[Команда reset 10](#_Toc72762496)

[Команда nearby 10](#_Toc72762497)

[Heroku сервіс. 13](#_Toc72762498)

[Висновки 14](#_Toc72762499)

[Список джерел інформації 15](#_Toc72762500)

# Вступ

Одна із сильних сторін месенджера Telegram - підтримка ботів, розумних помічників, які використовують технології штучного інтелекту і дозволяють вирішувати найрізноманітніші завдання.

Вперше платформа і програмний інтерфейс Bot API для створення ботів були представлені командою розробників Telegram в червні 2015 року. За цей час стараннями ентузіастів і професіоналів побачили світ понад 6 тисяч бот-сервісів для «Телеграма». Могутнім поштовхом до їх розвитку послужив оголошений в квітні Павлом Дуровим конкурс з сумарним призовим фондом в $ 1 млн. Ідея включення ботів в Telegram виявилася, поза всякими сумнівами, вдалим рішенням і припала до смаку не тільки для користувача аудиторії мессенджера і стороннім розробникам, а й бізнесу, отримав простий і зручний доступ до потужного інструменту для комунікацій з клієнтами в новому форматі.

# Необхідність телеграм боту.

Телеграм бот виконує функціонал розширеного блокноту який містить в собі список ресторанів, які користувач бажає відвідати в майбутньому. Наприклад, він йде по вулиці, бачить відмінний ресторан, і він хоче не забути відвідати його. Йому потрібно якимось чином відправити назву, адресу, та докласти інформацію.

У майбутньому, він зможе звернутися до боту і або просто переглянути список місць, які він раніше зберігав, або відправити поточне місце розташування і отримати адрес найближчого ресторану.

# Завдання

Для розробки бота буде необхідно зберігати дані. Також потрібно використовувати команди для пошукових робіт і обмін повідомлень з різним типом.

- Бот вітає користувача та надсилає повідомлення із текстом про команди, які він може виконувати.

- Бот реагує на команди:

- /аdd – відбувається покрокове введення адреси ресторану: спочатку назва (текст), далі надсилається фото ресторану, а опісля – його локація (location)

- /list – відображаються додані місця (надсилається назва, фото, локація кожного ресторану)

- /reset – видаляються додані місця

- Бот реагує на додаткову команду:

- /nearby - надсилає ресторани в радіусі 500 метрів при відправці локації, або повертає повідомлення про відсутність таких місць.

- Бот зберігає дані у нереляційній БД (Redis);

- Бот розміщений на сервісі Heroku та доступний у будь-який час, адже працює на віддаленому сервері.

# Практична частина

Спершу нам потрібно перейти до телеграму, знайти BotFather та зареєструвати там нашого бота.

Виконуємо дії як на рис. 1

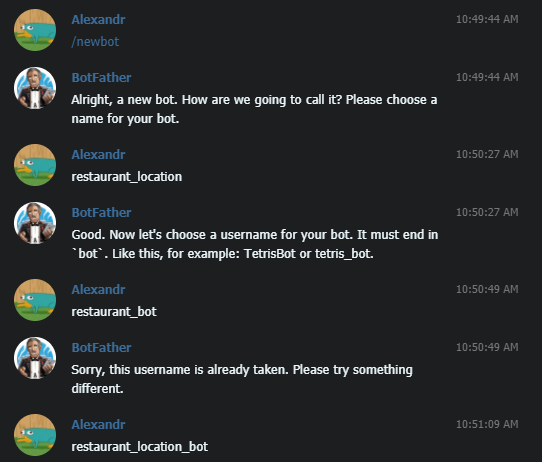


Рисунок 1 – Бесіда з ботом

Після цього бот видає нам **token** для роботи у пайтоні з нашим ботом.

Приступає до написання коду.

Спершу нам знадобиться імпортувати всі бібліотеки вказані нижче

import telebot # для роботи з телеграмом  
import requests # робити запит до гул карт  
import redis # не реляційна БД  
import os # для роботи з файлами(фотографіями\_

Далі створюємо об’єкти:

* Бот – передаємо **token**
* БД

bot = telebot.TeleBot(botKey)  
db = redis.Redis()

Ще нам залишається створити словник ресторанів, який буде містити інформацію в наступному вигляді **ключ: [список ресторанів].** Ключ буде у вигляді **id** який ми візьмемо із користувача який надішле повідомлення.

До речі щодо ресторанів, ось як виглядає клас **Ресторан:**

class Restaurant:  
 name: str  
 photo: str  
 location: dict  
  
 def \_\_init\_\_(self, name=None, photo\_path=None, location=None):  
 self.name = name  
 self.photo = photo\_path  
 self.location = location  
  
 def \_\_str\_\_(self):  
 return f"----------\n" \  
 f"Name: {self.name}\nPhoto: {self.photo}\nLocation: {self.location}"

Почнемо з найпростішого, з вітання.

@bot.message\_handler(commands=['start'])  
def send\_welcome(message):  
 bot.reply\_to(message, f"Hello, I'm restaurant bot. Nice to meet you, {message.from\_user.first\_name}\n\n"  
 f"This bot is being used for finding restaurant near you. \n\n"  
 f"Also you can use it as notebook of restaurants which you want to visit.\n\n"  
 f"/add - To add new restaurant. You need to send 3 messages: Name, Photo, Location\n"  
 f"/list - To show all restaurants\n"  
 f"/reset - To delete all restaurants\n"  
 f"/nearby - To show all restaurants near you within 500 meters")  
  
 print(message.from\_user)  
 restaurants[message.from\_user.id] = []

Як ми бачимо, для створення коду який реагує на команди від користувача потрібно використати вбудований декоратор у **telebot** під назвою **message\_handler,** та передаємо списочок команд, на які буде реагувати наша функція, в нашому випадку це **start.**

Наш бот **відповідає** на повідомлення користувача, довгим повідомленням з можливостями нашого боту.

Та у кінці функції створює елемент словнику з ключем **id** користувача, а значенням виступає список, поки що без жодних елементів. Але зараз ми їх добавимо.

## Команда add

Для додавання ресторанів потрібно написати декілька функцій, де введена інформація користувача буде послідовно оброблюватися та передаватися у наступну функцію, щоб у користувача не було можливості спочатку ввести фото потім локацію потім назву.

Тому для початку, при команді **add** запрошуємо у користувача ввести ім’я ресторану. Виглядає це наступним чином:

@bot.message\_handler(commands=['add'])  
def add\_request(message):  
 print('Add request started')  
 restaurants[message.from\_user.id].append(Restaurant())  
  
 send = bot.send\_message(message.chat.id, 'Enter Name of Restaurant')  
 bot.register\_next\_step\_handler(send, add\_name)

Спершу додаємо новий пустий об’єкт у список ресторанів, конкретного користувача. Відправляємо користувачу **повідомлення** “Enter Name of Restaurant”. Відповідь передаємо у наступну функцію за допомогою методу **register\_next\_step\_handler().**

Одним із параметрів являється **add\_name** це наша наступна функція яка запам’ятовує ім’я ресторану та запрошує наступні дані.

def add\_name(message):  
 print('Add name started')  
  
 try:  
 if message.text.lower() == 'cancel':  
 bot.reply\_to(message, 'Canceled')  
 del restaurants[message.from\_user.id][-1]  
 return  
 except Exception as e:  
 pass  
  
 try:  
 if message.text:  
 restaurants[message.from\_user.id][-1].name = message.text  
 else:  
 raise Exception  
  
 send = bot.send\_message(message.chat.id, 'Send me Photo of Restaurant')  
 bot.register\_next\_step\_handler(message, add\_photo)  
 except Exception as e:  
 send = bot.send\_message(message.chat.id, 'Enter Name of Restaurant')  
 bot.register\_next\_step\_handler(send, add\_name)

Функція приймає лише текст, якщо повідомлення його не містить то користувачеві дається змога повторно ввести ім’я ресторану. Якщо переданим текстом є слово **cancel** то дія припиняється. Пустий об’єкт, який був створений раніше, видаляється.

Далі ім’я зберігається, користувачу надсилається повідомлення “Send me Photo of Restaurant”, та фото яке передав користувач оброблюється у наступній функції **add\_photo.**

def add\_photo(message):  
 print('Add photo started')  
  
 try:  
 if message.text.lower() == 'cancel':  
 bot.reply\_to(message, 'Canceled')  
 del restaurants[message.from\_user.id][-1]  
 return  
 except Exception as e:  
 pass  
  
 try:  
 file\_id = message.photo[-1].file\_id  
 file = bot.get\_file(file\_id)  
 path = file.file\_path  
 downloaded = bot.download\_file(path)  
  
 filename = path.split('/')[1]  
  
 if not os.path.exists(str(message.chat.id) + '/'):  
 os.mkdir(str(message.chat.id) + '/')  
  
 photo\_path = str(message.chat.id) + '/' + filename  
  
 with open(photo\_path, 'wb') as new\_file:  
 new\_file.write(downloaded)  
  
 restaurants[message.from\_user.id][-1].photo = photo\_path  
  
 send = bot.send\_message(message.chat.id, 'Send me Location of Restaurant')  
 bot.register\_next\_step\_handler(message, add\_location)  
 except Exception as e:  
 send = bot.send\_message(message.chat.id, 'Send me Photo of Restaurant')  
 bot.register\_next\_step\_handler(message, add\_photo)

Функція приймає фотографії. Але ж знову є можливість відмінити команду текстом **cancel.**

Далі виконується алгоритм парсингу назви фотографії, компонується з **id** користувача. Потім скачується фото, і дані записуються у файл який в директорії під назвою **id** користувача, і ім’ям яке відповідає відправленої фотографії.

У об’єкт ресторану зберігається шлях до збереженої фотографії.

Далі викликається функція **add\_location**

def add\_location(message):  
 print('Add location started')  
  
 try:  
 if message.text.lower() == 'cancel':  
 bot.reply\_to(message, 'Canceled')  
 del restaurants[message.from\_user.id][-1]  
 return  
 except Exception as e:  
 pass  
  
 try:  
 print(message.location)  
 print(type(message.location))  
 if message.location:  
 restaurants[message.from\_user.id][-1].location = message.location  
 else:  
 raise Exception()  
 bot.send\_message(message.chat.id, 'Restaurant has been added in list')  
 except Exception as e:  
 send = bot.send\_message(message.chat.id, 'Send me Location of Restaurant')  
 bot.register\_next\_step\_handler(message, add\_location)  
  
 try:  
 db.rpush(message.from\_user.id,  
 restaurants[message.from\_user.id][-1].name,  
 restaurants[message.from\_user.id][-1].photo,  
 restaurants[message.from\_user.id][-1].location.latitude,  
 restaurants[message.from\_user.id][-1].location.longitude)  
 except Exception as e:  
 print(e)

Функція приймає об’єкт **локація.** Також є можливість відмінити команду.

Далі відбувається перевірка чи **локація** міститься у повідомленні. Якщо ні то виникає виключення яку приводить до повторного запиту у користувача даних.

Якщо інформація відповідає нормам, записується у об’єкт ресторану.

Оскільки це останній запит, то вкінці функції введена інформація зберігається у БД **Redis.**

## Команда list

Команда **list** оброблюється наступним чином:

@bot.message\_handler(commands=['list'])  
def show\_restaurants(message):  
 try:  
 if restaurants[message.from\_user.id]:  
 for i in restaurants[message.from\_user.id]:  
 bot.send\_message(message.chat.id, 'Restaurant Name: ' + i.name)  
  
 with open(i.photo, 'rb') as photo:  
 bot.send\_photo(message.chat.id, photo)  
  
 bot.send\_location(message.chat.id, i.location.latitude, i.location.longitude)  
 else:  
 bot.reply\_to(message, 'List is empty')  
 except Exception as e:  
 bot.reply\_to(message, "Error: Supposedly you have to send me /start for registration your ID")

Якщо існує об’єкт з таким користувачем, то програма проходиться по всім елементам списку цього користувача. І відправляє кожне поле поле **Ресторану** окремо. Якщо у такого користувача немає жодного **ресторану** то відправляється відповідне повідомлення.

Якщо такого користувача немає у БД то користувача просять ввести команду **start** для реєстрації його **id.**

## Команда reset

Спершу видаляє вся фотографії на хості. Потім словник ресторанів, видаляються дані у базі даних. Та відправляється повідомлення що список був очищен.

@bot.message\_handler(commands=['reset'])  
def del\_restaurants(message):  
 try:  
 for i in restaurants[message.from\_user.id]:  
 os.remove(i.photo)  
  
 os.rmdir(str(message.from\_user.id))  
  
 restaurants[message.from\_user.id].clear()  
 db.delete(message.from\_user.id)  
 bot.reply\_to(message, "List has reset")  
 except Exception as e:  
 bot.reply\_to(message, "Error: Supposedly you have to send me /start for registration your ID")  
 print(e)

Але якщо виникла помилка, то оброблюється лише той випадок коли не існує такого **id** користувача у списку.

## Команда nearby

Команда виконує дві послідовні функція, перша відправляє користувачу “Send me your location” .

@bot.message\_handler(commands=['nearby'])  
def nearby(message):  
 print("Nearby has started")  
  
 try:  
 if message.text.lower == 'cancel':  
 bot.reply\_to(message, 'Canceled')  
 return  
 except Exception as e:  
 pass  
  
 send = bot.send\_message(message.chat.id, "Send me your location")  
 bot.register\_next\_step\_handler(message, make\_request)

Отримане повідомлення від користувача передає функції **make\_requrst**

Ця функція робить запит на Google Maps API, запит вже містить параметри локації, радіус пошуку, та ключ, який ми отримали у Google Cloud Platform.

@bot.message\_handler(content\_types=['location'])  
def make\_request(message):  
 print("Make request has started")  
  
 try:  
 if message.text.lower() == 'cancel':  
 bot.reply\_to(message, 'Canceled')  
 return  
 except Exception as e:  
 pass  
  
 try:  
 print(message.location)  
 if message.location:  
 req = requests.get("https://maps.googleapis.com/maps/api/place/nearbysearch/json?location="  
 + str(message.location.latitude) + "," + str(message.location.longitude) +  
 "&radius=500&types=restaurant&key=" + googleApiKey)  
 j\_req = req.json()  
  
 results = j\_req['results']  
  
 for i in results:  
 name = i['name']  
 location = i['geometry']['location']  
  
 bot.send\_message(message.chat.id, "Name: " + name)  
 bot.send\_location(message.chat.id, location['lat'], location['lng'])  
 else:  
 raise Exception()  
 except Exception as e:  
 send = bot.send\_message(message.chat.id, "Send me your location")  
 bot.register\_next\_step\_handler(message, make\_request)

Щоб отримати ключ від Google Maps API потрібно провести реєстрацію у Google Cloud Platform. Там створити новий API та згенерувати ключ.

Після запиту парсимо .json файл. У списку **результатів** є поля з **name,** та з **geometry->location.** Зібрану інформацію відсилаємо користувачеві.

Також на всяк випадок, реалізуємо окрему функцію, яка читає БД

def read\_db():  
 for i in db.keys():  
 start = 0  
 end = 3  
  
 id = int(i.decode('utf-8'))  
 item = db.lrange(id, start, end)  
 restaurants[id] = []  
  
 while item:  
 start += 4  
 end += 4  
  
 name = item[0].decode('utf-8')  
 photo = item[1].decode('utf-8')  
 location = telebot.types.Location(float(item[3].decode('utf-8')), float(item[2].decode('utf-8')))  
  
 restaurants[id].append(Restaurant(name, photo, location))  
  
 item = db.lrange(id, start, end)

Виконується звичайне читання та парсинг. Виключаючи одне але. Оскільки список ресторанів це список в який елемент додаються в кінець, то два ресторани наприклад Bufet, Mafia. Виглядатимуть наступним чином:

1. Bufet
2. Path/to/photo/Bufet.png
3. LatitudeBufet
4. LongitudeBufet
5. Mafia
6. Path/to/photo/Mafia.png
7. LatitudeMafia
8. LongitudeMafia

То читання першого ресторану відбувається з 1-4 елемент списку. А другий ресторан з 5-8. У пайтоні список починається з нуля. Тому адаптуємо та парсимо запити.

## Heroku сервіс.

Ми на фінішній прямій, нам залишається залити нашого бота на сервіс. Для цього проходимо реєстрацію, створюємо проект. Далі потрібно створити два конфігураційних файла.

Файл **Procfile:** містить команду яку потрібно виконати на сервісі

worker: python main.py

Файл **requirements.txt:** містить бібліотеки які використовувались у програмі.

pyTelegramBotAPI==3.7.9  
requests==2.25.1  
redis==3.5.3

Далі виконуємо наступні команди у командній строчці:

$ heroku login

$ git add .

$ git commit -am "make it better"

$ git push heroku master

До речі нам знадобиться виконати команду

heroku config:get REDISCLOUD\_URL

Де відповідь виглядатиме наступним чином:

http://rediscloud:password@hostname:port

Нам знадобиться **hostname, password, port**

Тепер коли ми створювали у пайтоні

db = redis.Redis()

Заміняємо на:

db = redis.Redis(host=**hostname**, port=**port**, password=**password**)

На своєму аккаунті Heroku вмикаємо програму, та додаємо **add-on(**Рис. 2**)**

****

**Рисунок 2 –** Add-on

Вітаю, проект готовий!

# Висновки

Відповідно з поставленим завданням були отримані наступні результати:

- Бот вітає користувача та надсилає повідомлення із текстом про команди, які він може виконувати.

- Бот реагує на команди:

- /аdd – відбувається покрокове введення адреси ресторану: спочатку назва (текст), далі надсилається фото ресторану, а опісля – його локація (location)

- /list – відображаються додані місця (надсилається назва, фото, локація кожного ресторану)

- /reset – видаляються додані місця

- Бот реагує на додаткову команду:

- /nearby - надсилає ресторани в радіусі 500 метрів при відправці локації, або повертає повідомлення про відсутність таких місць.

- Бот зберігає дані у нереляційній БД (Redis);

- Бот розміщений на сервісі Heroku та доступний у будь-який час, адже працює на віддаленому сервері.

Програма запущена та перевірена на помилки.

[Посилання на **github** проекту.](https://github.com/AlexandrKushch/python)

# Список джерел інформації

1. <https://core.telegram.org/bots/api>
2. <https://pypi.org/project/pyTelegramBotAPI/0.3.0/>
3. <https://developers.google.com/maps/documentation/distance-matrix/intro>
4. <https://id.heroku.com/login>
5. <https://pypi.org/project/redis/>
6. <https://pypi.org/project/requests/2.7.0/>